박사 학위논문

Ph. D. Dissertation

몬테카를로 광선 추적법을 위한 가속화 기술

Acceleration Techniques for Monte Carlo Ray Tracing

문 보 창 (文 普 昶 Moon, Bochang)

전산학과

Department of Computer Science

KAIST

2014

몬테카를로 광선 추적법을 위한 가속화 기술

Acceleration Techniques for Monte Carlo Ray Tracing

Acceleration Techniques for Monte Carlo Ray Tracing

Advisor : Professor Sung-Eui Yoon

by

Moon, Bochang

Department of Computer Science

KAIST

A thesis submitted to the faculty of KAIST in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the Department of Computer Science . The study was conducted in accordance with Code of Research Ethics[1].

2014. 05. 23.

Approved by

Professor Sung-Eui Yoon

[Advisor]

[1]Declaration of Ethical Conduct in Research: I, as a graduate student of KAIST, hereby declare that I have not committed any acts that may damage the credibility of my research. These include, but are not limited to: falsification, thesis written by someone else, distortion of research findings or plagiarism. I affirm that my thesis contains honest conclusions based on my own careful research under the guidance of my thesis advisor.

몬테카를로 광선 추적법을 위한 가속화 기술

문 보 창

위 논문은 한국과학기술원 박사학위논문으로
학위논문심사위원회에서 심사 통과하였음.

2014년 05월 23일

심사위원장 윤 성 의 (인)

심사위원 김 경 국 (인)

심사위원 김 민 혁 (인)

심사위원 박 진 아 (인)

심사위원 이 성 길 (인)

DCS
20105068

문 보 창. Moon, Bochang. Acceleration Techniques for Monte Carlo Ray Tracing. 몬테카를로 광선 추적법을 위한 가속화 기술. Department of Computer Science . 2014. 61p. Advisor Prof. Sung-Eui Yoon. Text in English.

ABSTRACT

Monte Carlo (MC) ray tracing has been considered as the most effective technique to produce a variety of realistic visual effects. However, its performance tends to be very slow since a lot of ray samples should be generated until we achieve converged images. In this thesis, we propose three novel techniques to accelerate performance of MC ray tracing. We fist develop a ray reordering framework based on a novel ray ordering measure *hit point heuristic* to compute a cache-coherent access pattern of ray traversals on acceleration hierarchies. In addition to the cache optimization technique, we propose an efficient and robust image-space denoising method for reducing noise generated by MC ray tracing while preserving image features. Our denoising is built upon a novel edge-stopping function *virtual flash image* which captures a wide variety of image features without taking additional ray samples. Furthermore, we present a new image-space adaptive rendering method based on locally weighted regression. In our adaptive framework, we locally guide ray budgets on high error regions, and estimate optimal filtering bandwidths for each rendering feature in terms of minimizing filtering errors. We have demonstrated that the proposed acceleration techniques improve the performance of MC ray tracing using different realistic benchmarks compared to state-of-the-art methods.

Contents

List of Tables

List of Figures

Chapter 1. Introduction

Photo-realistic rendering is one of long-standing problems in computer graphics, and Monte Carlo (MC) ray tracing such as path tracing [33] and photon mapping [32] has been widely applied for synthesizing physically accurate rendering images. This is mainly because MC ray tracing is general and effective for synthesizing a variety of rendering effects such as soft shadows, reflections, caustics, motion blur, and etc. Unfortunately, it typically requires a lot of ray samples such as primary, secondary, and shadow rays until we reach to a converged image. The intrinsic nature of MC ray tracing often results in its performance degradation, and the problem has been recognized as one of its main challenges.

As one way to accelerate MC ray tracing, the algorithms for improving the performance of its core algorithm *ray tracing* have been developed. Examples of the techniques are designing efficient intersection tests, constructing acceleration hierarchies, and exploiting data level parallelism using the SIMD functionality and GPUs [65, 53, 79].

Most research in this direction aims at improving performance of ray tracing with primary rays, but processing secondary rays in an efficient way has received much attention in recent years. However, it is widely known that secondary rays generated for simulating global illumination effects show a low ray coherence and thus a low cache utilization during processing those rays. One of main challenges is therefore achieving a high cache utilization during the secondary ray processing.

The well-known approaches of improving cache utilization for ray tracing can be classified as two categories: layout reordering and ray ordering. Layout reordering techniques [62, 84] aims at generating cache-coherent mesh or hierarchy layouts in main memory or external drives. Achieving higher cache utilization can be achieved with cache-coherent ray access pattern introduced by ray reordering methods [52, 47, 6] as well as the layout reordering. Existing studies on ray reordering have mainly focused on either reducing L1/L2 caches for small models or reducing the disk I/O accesses for out-of-core models. These approaches can compute cache-coherent accesses, but the complexity of existing ray tracing systems increases due to coupling ray traversal and ray reordering modules.

In this thesis, we propose a cache-oblivious ray reordering method that improves cache utilization for different cache levels such as L1/L2 caches or main memory (Chapter 3). We decouples our ray ordering module from existing ray tracing systems so that implementation efforts required to integrate reordering on existing ray tracing can be minimized. In addition to the decoupling, we propose a new ray reordering measure *hit point heuristic* which can be defined as intersection points between rays and models. We have demonstrated that the ray tracing with our reordering method shows an order of magnitude performance improvement compared to rendering without reordering.

The cache coherent ray processing achieved by reordering approaches can lead to high performance improvements of MC ray tracing, but the performance can be still too slow since it requires a lot of ray samples until we reach to converged images. Instead of processing more rays to generate smooth image, we can directly remove noise in the images generated by a small number of ray samples as an acceleration approach for MC ray tracing. The image denoising has been a popular approach thanks to its effectiveness and simplicity, but still remains as a challenging problem since distinguishing image features (i.e., edges) from noise is fundamentally difficult.

We present a novel edge-stopping function *virtual flash images* which guides image filtering methods

so that edges introduced by different rendering effects can be well preserved (Chapter 4). We create the virtual flash image by considering a subset of light paths (e.g., direct illumination) with an additional point light that emulates a camera flash. We have demonstrated an existing denoising method guided by the virtual flash images outperforms previous denoising methods in terms of visual quality and numerical accuracy.

Image denoising approaches can be a simple and effective means for noise reduction in the images rendered by MC ray tracing, but it has been commonly considered that the image denoising produces systematic errors (i.e., bias) while reducing random errors (i.e., variance). Adaptive rendering methods to minimize both errors have a long history [81, 46], and its key component is an error estimation process to guide both sample allocation for allocating more samples on high error regions and select optimal filtering bandwidths.

We classify adaptive rendering approaches into integrand- and image-space approaches. Integrand-space methods [26] can be a powerful way to reduce the required number of ray samples until we reach to smooth images, but image-space techniques have been more focused due to its efficiency and simplicity for integration into existing rendering systems. Furthermore, the image-space methods can handle a wide variety of rendering effects simultaneously.

We present a new image-space adaptive rendering method based on local weighted regression, and also propose an error analysis by utilizing well-established local regression theory (Chapter 5). Our key contribution is that our method support anisotropic bandwidth selection for different rendering features such as normals, textures, and depth. In addition to the bandwidth selection, our adaptive sampling is designed based on the proposed error analysis in a principled way. We have demonstrated that the proposed approach outperforms the state-of-the-art methods [60, 41] in terms of visual quality and numerical accuracy.

1.1 List of Related Papers

This thesis is partially related to the following published papers:

- **Bochang Moon**, Nathan Carr, Sung-Eui Yoon, *Adaptive Rendering based on Weighted Local Regression*, Accepted in ACM Transactions on Graphics, 2014.
- **Bochang Moon**, Jong Yun Jun, JongHyeob Lee, Kunho Kim, Toshiya Hachisuka, Sung-Eui Yoon, *Robust Image Denoising using a Virtual Flash Image for Monte Carlo Ray Tracing*, Computer Graphics Forum, vol. 32, no. 1, pp. 139-151, 2013.
- **Bochang Moon**, Youngyong Byun, Tae-Joon Kim, Pio Claudio, Hye-sun Kim, Yun-ji Ban, Seung Woo Nam, Sung-Eui Yoon, *Cache-Oblivious Ray Reordering*, ACM Transactions on Graphics, Vol. 29, No. 3, 2010.

Chapter 2. Background and Related Work

2.1 Monte Carlo Ray Tracing

Monte Carlo (MC) ray tracing [33, 32] has been recognized as a powerful approach to handle realistic rendering effects as shown in Fig. 2.1.

Figure 2.1: Monte Carlo ray tracing results with different benchmarks.

At a high level, MC ray tracing can be considered as an efficient way to solve the following rendering equation [33]:

$$L_o(\mathbf{x}, \vec{\omega}_o) = L_e(\mathbf{x}, \vec{\omega}_o) + \int_S f_r(\mathbf{x}, \vec{\omega}_i, \vec{\omega}_o) L_i(\mathbf{x}' \rightarrow \mathbf{x}) \delta(\mathbf{x}, \mathbf{x}') \frac{(\vec{\omega}_i \cdot \vec{n}')(\vec{\omega}_i \cdot \vec{n})}{||\mathbf{x} - \mathbf{x}'||^2} dA' \tag{2.1}$$

where $L_o(\mathbf{x}, \vec{\omega}_o)$ is the outgoing or reflected radiance at $\mathbf{x}$ with direction $\vec{\omega}_o$. $L_e(\mathbf{x}, \vec{\omega}_o)$ and $L_i(\mathbf{x}' \rightarrow \mathbf{x})$ are the emitted radiance and incoming radiance from $\mathbf{x}'$ to $\mathbf{x}$ respectively, and $f_r(\mathbf{x}, \vec{\omega}_i, \vec{\omega}_o)$ the bidirectional reflectance distribution function (BRDF). $\delta(\mathbf{x}, \mathbf{x}')$ is the Dirac delta function that takes one only if $\mathbf{x}$ and $\mathbf{x}'$ are mutually visible. Otherwise, the function takes zero.

The rendering techniques for solving the equation 2.1 have been studied over past decades. In the radiosity method [23], surfaces in scenes are typically discretized into patches, and the energy between the patches are computed based on the assumption that all the surfaces have Larmbertian reflectance. The approach can be an effective means to synthesize some global illumination effects such as diffuse interreflections (e.g., color bleeding). Whitted [82] proposed an illumination model which considers specular reflections and refractions, and a ray tracing was presented to determine visible surfaces. Cook et al. [11] introduced a Monte Carlo ray tracing approach which simulates distributed rendering effects such as soft shadows, motion blur, and depth of field effects. In 1986, Kajiya [33] presented the rendering equation 2.1 and also its general solution, *path tracing* that synthesizes a variety of global illumination effects. The path tracing is commonly considered as an unbiased Monte Carlo ray tracing that does not introduce any systematic errors (i.e., bias). Jensen [32] proposed a two-pass rendering algorithm, *photon mapping* that robustly handles difficult light paths (e.g., caustics) by using a density estimation of photons.

At a high level, MC ray tracing methods [11, 33, 32] traces randomly generated rays and averages the radiance values computed at intersection points between rays and models in order to approximate the

integral in eq. 2.1. The MC ray tracing is a conceptually simple and general approach, but suffers from slow convergence. For example, the results of MC ray tracing often shows random artifacts (i.e., noise) unless a large number of ray samples are not used. In addition to the slow convergence, the processing time of finding intersection points can be large especially when we process secondary rays with a low ray coherence. For example, the generated rays in MC ray tracing should traverse acceleration hierarchies and meshes stored in a main memory or disks in order to find intersection points between the rays and models. The rays with a low ray coherence randomly access the data stored in the memory, and it can lead to a lot of L1/L2 cache misses and disk I/O.

2.2 Reordering

Ray tracing and global illumination methods have been well studied. Also, good surveys and books are available [65, 53, 79]. In this section, we review prior work related directly to our problem.

2.2.1 Computation Reordering

Computation reordering strives to achieve a cache-coherent order of runtime operations in order to improve program locality and reduce the number of cache misses. Computation reordering methods can be classified into either *cache-aware* or *cache-oblivious*. Cache-aware algorithms utilize the knowledge of cache parameters, such as cache block size [76]. On the other hand, cache-oblivious algorithms do not assume any knowledge of cache parameters [21]. There is a considerable amount of literature on developing cache-efficient computation reordering algorithms for specific problems and applications [2, 76]. In computer graphics, out-of-core algorithms [66], which are cache-aware methods, have been designed to handle massive models.

2.2.2 Cache-Coherent Ray Tracing

There has been extensive research on exploiting the coherence in ray tracing. These can be classified into packet methods, layout reordering, and ray reordering methods.

Packet ray tracing: Neighboring rays can exhibit spatial coherence and utilizing this coherence can improve the performance of ray tracing. Earlier attempts include beam tracing [30]. Wald et al. [80] exploited the coherence of primary and shadow rays by grouping rays into packets and utilizing the SIMD functionality of modern processors. Reshetov *et al.* [57] proposed an algorithm to integrate beam tracing with the kd-tree spatial structure and were able to further exploit coherence of primary and shadow rays. There have been a few ray reordering methods that can utilize the SIMD functionality for secondary rays [4, 24]. These ray reordering methods for the SIMD utilization can be performed on rays reordered by our method.

Layout reordering: The order of data stored in memory or external drives can affect the performance of ray tracing, given the widely used block-fetching caching scheme [84]. In this caching scheme, blocking related nodes in a cluster can reduce the number of cache misses. The *van Emde Boas* layouts of trees [75] are constructed by performing a recursive blocking to nodes. Havran analyzes various layouts of hierarchies in the context of ray tracing and improves the performance by using a compact layout representation of hierarchies [28]. Yoon and Manocha [87] developed cache-efficient layouts of hierarchies

for ray tracing. Also, there are a few cache-coherent mesh layouts [86, 85, 62].

Ray reordering: To reorder primary rays, space-filling curves like Z-curves [62] have been used. Mansson et al. [44] showed coherence among secondary rays based on their proposed ray coherence measures. However, it was not demonstrated to achieve a higher runtime performance based on their proposed ray reordering heuristics. Pharr et al. [52] proposed a ray reordering method for ray tracing massive models that cannot fit into main memory. Their method uses a scheduling grid for queueing rays and processes rays in a coherent manner, while considering the available cache information. Steinhurst et al. [68] reorder kNN searches of photon mapping to reduce the memory bandwidth. Navratil et al. [47] presented a ray scheduling approach that improves a cache utilization and reduces DRAM-to-cache bandwidth usage. Budge et al. [6] employed a ray reordering method to utilize hybrid resources such as multiple CPUs and GPUs. These techniques are based on Pharr et al.'s ray reordering method, which couples the ray reordering and the scene traversal. By doing so, these methods can easily know which parts of meshes and hierarchies are accessed and cached during processing of rays. A downside of these techniques is that by coupling the ray reordering and scene traversal, the modularity of these methods is lowered.

2.2.3 Ray Tracing Massive Models

Ray tracing massive models has been studied well. In-core techniques exist to perform the ray tracing of massive datasets [14, 69] by using large, shared memory systems. There are also out-of-core techniques including latency hiding [78]. There are different approaches aiming at designing compact representations, by applying the quantization on acceleration hierarchies [9], reducing costs of representing meshes and hierarchies [39, 36], or efficient culling techniques [56]. These methods can be combined with our proposed method to further improve the performance of ray tracing massive models.

2.3 Noise Reduction

2.3.1 Noise Reduction for Monte Carlo Ray Tracing

Image space reconstruction methods. Image filtering has been a popular approach to remove noise in images generated by MC ray tracing, because of its simplicity and efficiency. The well-known image filters in the image processing field [50, 74, 17] can be applied to rendered images, but have been often tailored to rendering. It has been recognized that geometric information can play an important role for predicting edges in rendered images. For example, McCool [45] proposed the use of an anisotropic diffusion process guided by geometric features stored at the G-buffer for removing noise.

Interactive rendering techniques also used different types of filtering that commonly uses geometric features for quickly producing reasonable image quality. They used the cross bilateral filters [58], guided image filter [3], Á-Trous wavelet transform [13], and filtering stochastic buffers [64].

Sen et al. [63] recently showed that by analyzing the functional relationships between MC inputs and outputs, feature weights can be computed to achieve very high quality image reconstructions with a low number of samples. While their proposed method of *random parameter filtering* (RPF) produces exceptional reconstruction results, it comes at a high computation and storage cost. Because of these issues, RPF was not demonstrated for adaptive rendering. In contrast our method is designed to have both a low storage and computation overhead, while producing better filtered results efficiently.

Our method departs from earlier work in that it is derived from local regression theory appearing in the statistics literature [8, 61]. We reconstruct a surface, allowing us to efficiently estimate bias, variance, and partial derivatives, which play a crucial role in our error analysis. This in turn allows us to select ideal filter widths across multiple feature dimensions. Local regression has been used as a tool for various image processing [72]. The fundamental difference between our work and them is that our method is tailored for considering additional information available for rendering and ignoring noisy inputs, while local regression used in the image processing field assumes that input vectors do not contain any noise. Furthermore, our method is equipped with robust error analysis for a tight coupling with the sampling process.

Multi-dimensional reconstruction and sampling. Numerous methods have been proposed to operate in integrand space, where reconstruction and sampling is performed in high dimensions. Hachisuka et al. [26] proposed a general multi-dimensional adaptive rendering method that reconstructs smooth images using Riemann sums. This method works quite well in a low dimensional space, however its effectiveness rapidly degrades as the dimension increases. Similar approaches have been proposed to examine a reduced set of effects to mitigate the so call *curse of dimensionality*. Soler et al. [67] analyzed the depth-of-field effect in the frequency domain, and adaptively varied sample density over image and lens domains. Egan et al. [18] designed sheared reconstruction filters based on frequency analysis in order to efficiently render motion blur and soft shadows. Lehtinen et al. [40] proposed a reconstruction method for simultaneously synthesizing depth-of-field, motion blur, and soft shadows, while reusing samples in a visibility-aware manner.

While multi-dimensional reconstruction and sampling methods have shown exceptional performance, both the computational and storage costs can reduce their effectiveness in practice. The generality of our method allows us to work across a wide range of effects. In addition, we are able to determine an appropriate, reduced dimensional feature subspace locally, avoiding an unnecessary growth in dimension and improving efficiency.

Image space adaptive rendering. We now detail the techniques that are most closely related to our method. Overbeck et al. [48] developed a framework that treats sampling and reconstruction as a coupled iterative process. They decompose the image into wavelets and apply shrinkage to the coefficients to reduce noise. By doing so they were able to achieve high quality images with relatively few samples. The same iterative framework has been adopted by more recent works [59, 60, 41], and is also a foundation upon which we build our research.

Rousselle et al. [59] demonstrated improved image quality by greedily selecting an appropriate isotropic filtering bandwidth locally across the image. This work was further improved by introducing state-of-the-art anisotropic filtering methods such as the cross bilateral and non-local means filters [60, 41]. Li et al. [41] introduced the use of Stein's unbiased risk estimator for sampling and bandwidth selection. Using this estimator their system is capable of supporting a wide range of anisotropic filters including the cross bilateral and non-local means. One limitation of their work is that their bandwidth selection process is over the spatial dimension and relies on a set of fixed *global* parameters for controlling the influence of other geometric features such as depth, texture, and normals. As shown by Sen et al. [63] analyzing the functional relationship between the geometric feature information and output intensity can significantly improve image quality. Our work addresses this problem by performing an optimal bandwidth selection locally for different feature types.

2.3.2 Noise Reduction for Photographs

In the field of image processing, commonly used techniques adopt an edge-preserving filter for denoising photographs. Well-known filters include anisotropic diffusion [50] and bilateral filtering [74]. Wavelet-based methods denoise images by thresholding the wavelet coefficients [17]. Wavelet-based methods, however, can produce distracting image artifacts such as low-frequency noise and edge ringing caused by underlying wavelet basis. Unfortunately, direct applications of such techniques to denoise rendered images have shown sub-optimal results, since they do not utilize various information available during the rendering process.

Image enhancement by flash photography: Eisemann and Durand [19] and Petschnigg et al. [51] designed an effective denoising method for photographs taken in dark environments, by utilizing additional photographs taken with a camera flash. They extended bilateral filtering into a cross (or joint) bilateral filtering that considers pairs of flash and non-flash images. The key observation of these methods is that the flash image is relatively sharp and less noisy compared to its corresponding non-flash image. Therefore, they could use the flash image as an estimator of the high-frequency content of the non-flash image. Inspired by these techniques, virtual flash images are designed for denoising rendered images while preserving various image features generated by Monte Carlo ray tracing methods.

Chapter 3. Cache-Oblivious Ray Reordering

3.1 Overview

In this section, we discuss the ray coherence of different types of rays and briefly explain the overall approach of our method.

3.1.1 Ray Coherence

Ray tracing generates a lot of rays to simulate various visual effects. These rays can be classified as primary, shadow, and secondary rays. Primary rays are known to show a high coherence during the hierarchy traversal and mesh accesses. Space-filling curves such as Z-curves have been used to reorder primary rays [52], based on positions of primary rays in the image plane. Once a primary ray has intersected with an object, shadow rays to lights and secondary rays (e.g., reflection rays), depending on the material property of the intersected object, are generated. Since light positions can be arbitrary and the intersected geometry can have an arbitrary normal, shadow and secondary rays generally have a lower coherence than primary rays. If rays are incoherent, then the data access pattern on the acceleration hierarchies and meshes can be incoherent. This incoherence may result in a high number of cache misses in various memory levels and lower the runtime performance. Therefore, processing rays in a cache-coherent manner is critical to design cache-coherent ray tracers.

3.1.2 Ray Reordering Framework

In order to reorder rays, we use a ray reordering framework (see Fig. 3.1) extended from typical ray tracing systems. This framework consists of ray generation, ray reordering, and ray processing modules. The ray generation module constructs rays including primary, secondary, and shadow rays. The ray processing module takes each ray and finds a hit point between the ray and the scene by accessing acceleration hierarchies and the meshes of the scene. Also, the ray processing module performs shading based on the hit point and its corresponding material information. If we have to generate shadow and secondary rays, the ray processing module sends the hit points and material information to the ray generation module. Typical ray tracing systems consist of only these two modules and process rays as they are generated without reordering rays.

In addition to these modules, we also use the ray reordering module. The ray reordering module maintains a *ray buffer* that can hold a user defined number of rays. Once the ray generation module constructs rays, these rays are stored in the ray buffer and then reordered in a way such that meshes and hierarchies are accessed in a cache-coherent manner during processing of reordered rays in the ray processing module. Note that our ray reordering framework is similar to previous ray reordering methods [52, 47, 6]. The main difference of our framework over these prior methods is that we decouple the ray reordering module from other modules, thereby achieving high modularity.

Given this ray reordering framework, the key component that governs the performance improvement is the ray reordering method. To maximize the benefits of the reordering method, the overhead of reordering should be kept small. We propose a simple cache-oblivious reordering method that has a low

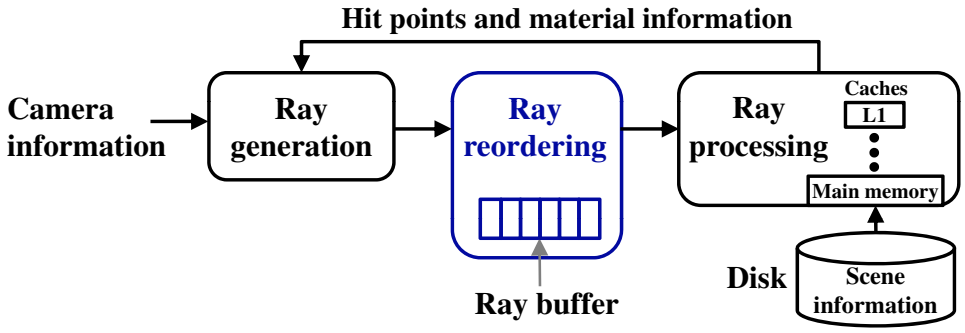


Figure 3.1: This figure shows different modules of our ray reordering framework. Our main contribution is the hit point heuristic (HPH) based ray reordering method employed in the ray reordering module.

reordering overhead, increases the cache coherence, and improves the performance of ray tracing models that have different model complexities.

Cache-coherent layouts of meshes and hierarchies: Our ray reordering method works on the assumption that geometrically close mesh data (e.g., vertices or triangles) and topologically close hierarchy data (e.g., nodes) are also stored closely in their corresponding mesh and hierarchy layouts respectively. There are many layouts satisfying such a property for meshes [62, 15, 85] and for hierarchies [75, 28, 87]. In our implementation, we use cache-oblivious layouts of meshes and hierarchies [85, 87]

3.2 Cache-Oblivious Ray Reordering

In this section we introduce our cache-oblivious ray reordering method.

3.2.1 Hit Point Heuristic

To reorder rays, we propose a *hit point heuristic* (HPH). A hit point of a ray is defined as the first intersection point computed between the ray and the scene, starting from the ray's origin. The main idea of the HPH method is to reorder rays based on their hit points using a space-filling curve (e.g., Z-curve).

The rationale why we use the hit point of a ray as a reordering measure is twofold. First, if the hit points of rays are geometrically close to each other, then the mesh regions accessed during processing of these rays are likely to be close too. Second, suppose that a hierarchy is decomposed into lower and upper regions. Lower regions of the hierarchy are closer to leaf nodes and upper regions of the hierarchy are closer to the root node of the hierarchy. Then, the lower regions of the hierarchy accessed during processing of rays whose hit points are close are likely to be close too because of the same reason that were for meshes (see Fig. 3.3). Although hit points of rays are close to each other, these rays' directions may be very different. In this case, their access patterns on upper regions of the hierarchy may be very different (see Fig. 3.3-(b)). However, the size of these upper regions of the hierarchy is relatively small compared to those of lower regions of the hierarchy. Also, the upper regions of the hierarchy are accessed by almost all the rays and thus are unlikely to be unloaded from the cache. Therefore, we may not get additional cache misses during processing of rays with the upper regions of the hierarchy.

To empirically verify the second rationale, we simulate a 6MB wide 24-way set-associative L2 cache of our test machine and measure L2 cache misses that occur in the upper and lower regions of a hierarchy

Figure 3.2: Photon mapping of an Armadillo (346 K triangles and 43.5 MB) in the Cornell box.

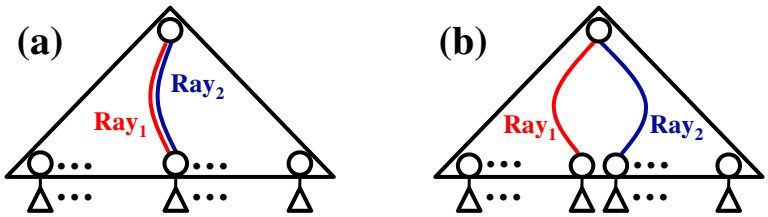


Figure 3.3: These two figures show data access patterns on the hierarchy during processing of two different rays, whose hit points are close to each other. The difference between the left and right figures is that two rays' directions are similar in the left, but different in the right.

during photon mapping of the Armadillo model in the Cornell box scene (Fig. 3.2). The number of L2 cache misses occurring in the lower regions of the hierarchy is significantly higher (e.g., 141.4 times higher than that occurring in the upper region of the hierarchy. As a result, we conclude that hit points between rays and the scene are equally or more important features to our problem than ray directions and ray origins, which have been widely considered as reordering measures in most prior works.

3.2.2 Approximate Hit Points

An issue of the HPH method is that it requires hit points between rays and the scene to reorder rays. However, computing these hit points requires processing of rays by traversing the hierarchy and accessing the mesh, which may cause a high number of cache misses that we attempted to avoid by reordering. To address this problem, we compute approximate hit points efficiently by performing the intersection tests between rays and simplified representations of the original models.

We compute a simplified representation of the original model using an out-of-core mesh simplification method [84]. This simplification method decomposes an input model into a set of clusters, each of which can be stored in main memory. Then, we simplify each cluster one by one. In order to compute approximate hit points that are close to the exact hit points, the simplified model should be geometrically similar to the original model. We use quadrics and choose edge collapses in an increasing order of simplification errors for each cluster by using a heap [22] within each cluster. While simplifying each cluster, we also allow simplifying edges that span multiple clusters. For a simplified representation, we set the bounding box of the simplified model to be the bounding box of the original model. Therefore, if a ray does not intersect with the bounding box, it is guaranteed that the ray does not intersect with

the original model.

Although simplification techniques including ours that rely on quadrics and edge collapses have been known to work well for various polygonal models [43], we found that it does not work well with models with lots of small objects including a furry squirrel model (shown in the right image of Fig. 3.5) in our benchmark models. Fortunately, we found that a recent stochastic simplification technique [10] works quite well for such models that have aggregate detail. We use this stochastic simplification method only for the furry squirrel model, given our out-of-core simplification framework described above.

For each simplified representation, we build a hierarchy in the same manner as building the hierarchy for the original model. In order to reduce the overhead of computing hit points with the simplified representations at runtime, we drastically simplify the models. In our tests, we use simplified models consisting of 2% of the complexity of the original models. We found that this strikes a good balance between the overhead of our method and the approximation quality and thus achieves the best performance improvement of using our ray reordering method (see Sec. 3.3.3).

To compute approximate hit points of rays, we perform intersection tests between the rays and the simplified models of the scene. If a ray intersects with one primitive of the simplified models, we use the hit point for the ray reordering. If the ray does not intersect with any primitives of simplified models, but one of the bounding boxes of the original models, we use the intersection point between the ray and the bounding boxes as a *virtual hit point* and use it for the ray reordering. For other rays that do not intersect with any of the bounding boxes, we terminate the processing of these rays, since it is guaranteed that they do not intersect with the original models of the scene. We use the computed approximate hit points only for reordering, not for other computations (e.g., shading).

One may consider to use virtual hit points as approximate hit points even for rays intersected with the scene, instead of using high quality simplified representations. However, we found that using only virtual hit points produces rather low-quality approximation results and using high quality simplified representations shows much higher (e.g., up to 12.1 times) performance improvements in our benchmark scenes.

3.2.3 Space-Filling Curve based Reordering

Once we compute approximate hit points for rays stored in the ray buffer, we reorder these rays by using a Z-curve, a simple space-filling curve. Since a Z-curve is defined in a uniform structure, we place hit points in a grid and compute ordering keys for these hit points by using a Z-curve ordering of cells in the grid structure.

We define the grid to enclose the bounding volume of the scene and to have $2^k \times 2^k \times 2^k$ cells. Then, we quantize each of three coordinates of a hit point into a k-bit integer. It has been known that the z-curve ordering key of such a point is computed by simply interleaving bits of three k-bit integers of the quantized three coordinates of the point [37]. For example, suppose that $x_k \cdots x_1$, $y_k \cdots y_1$, and $z_k \cdots z_1$ are three k-bit integers of the quantized three coordinates. Then, the z-curve ordering key of such point is defined by a 3k-bit integer of $x_k y_k z_k \cdots x_1 y_1 z_1$. An example of the Z-curve ordering keys of hit points is shown in Fig. 3.4.

In our current implementation, we choose k to be 20. Therefore, the ordering key for each hit point is represented with 60 bits that can be stored in an 8-byte integer. Also, our grid structure decomposes the bounding volume of the scene into 2^{60} uniform-sized cells. Therefore, most final ordering keys computed from rays are likely to be unique with models that we can have in practice. We also tried Hilbert-curves [62], but found that Z-curves are easier to implement and have less computations, while having

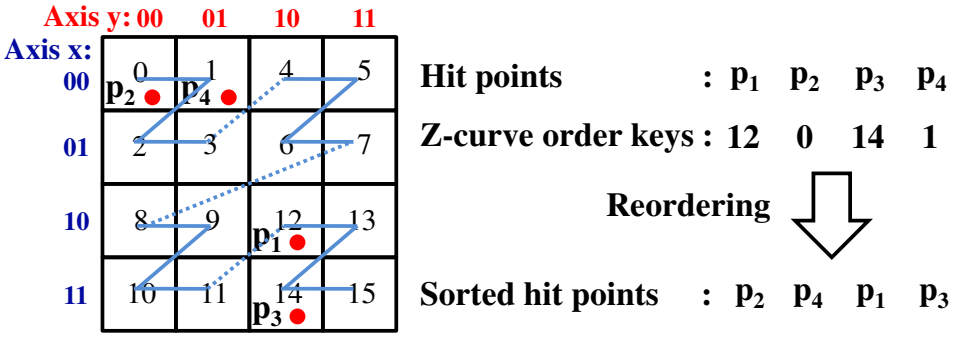


Figure 3.4: This figure shows an ordering of hit points with the Z-curve ordering of cells in the uniform grid.

only minor performance degradation (e.g., 2%) over Hilbert-curves.

Once we compute the ordering keys for rays, we sort rays based on the ordering keys. We use the 2-way merge sort due to its simplicity. After sorting rays using their associated approximate hit points, sorted rays are processed in the ray processing module.

3.3 Implementations and Results

We have implemented and integrated our ray reordering module in a CPU-based out-of-core ray tracing system. Our ray tracing system uses bounding volume hierarchies (BVHs) with axis-aligned bounding volumes for models [77, 38]. Also, to design an out-of-core ray tracing system, we employ an out-of-core data access framework for meshes and hierarchies [35]. This framework maintains a memory pool that consists of pages, each of which holds 4 MB of data. The size of the memory pool is determined by the available main memory. We also employ a simple memory management method based on the least-recently used (LRU) replacement policy. To implement the LRU replacement policy, we maintain a LRU list containing pages that have been accessed during the mesh and hierarchy traversal for ray tracing.

We use 512 by 512 image resolutions and perform various tests with a 32 bit Windows machine consisting of a 3.0 GHz processor, a disk that supports a sequential reading performance of 101 MB per second, and 4 GB memory, unless mentioned otherwise. Although the machine has 4 GB main memory, all the programs in the 32 bit Windows can use only up to 3.25 GB. Also, the Windows OS in our test machine uses about 0.2 GB. Therefore, our ray tracer can use up to about 3.05 GB and we use this as the maximum size of the memory pool for our out-of-core data access framework.

Ray processing throughputs: Our out-of-core ray tracer does not have a high ray processing throughput that is comparable to those of the-state-of-the-art ray tracers. When we test our ray tracer generating only primary rays with small models (e.g., Stanford bunny) that fit into main memory, our single-threaded ray tracer can process 1 million rays per second. Also, when we test our ray tracer for path tracing the Sponza scene with enough main memory (e.g., 16 GB) that can hold all the data, our ray tracer can process 82.3 K rays per second; detailed rendering configurations will be given in Sec. 3.3.1. Our method uses out-of-core abstractions, which have high overheads. Also, our method does not use any packet tracing methods; if we implement recent packet tracing methods, we expect that our ray tracer can have higher ray processing throughputs.

Ray buffer: Our ray buffer consists of in-core and out-of-core parts. We allocate only 88 MB of the

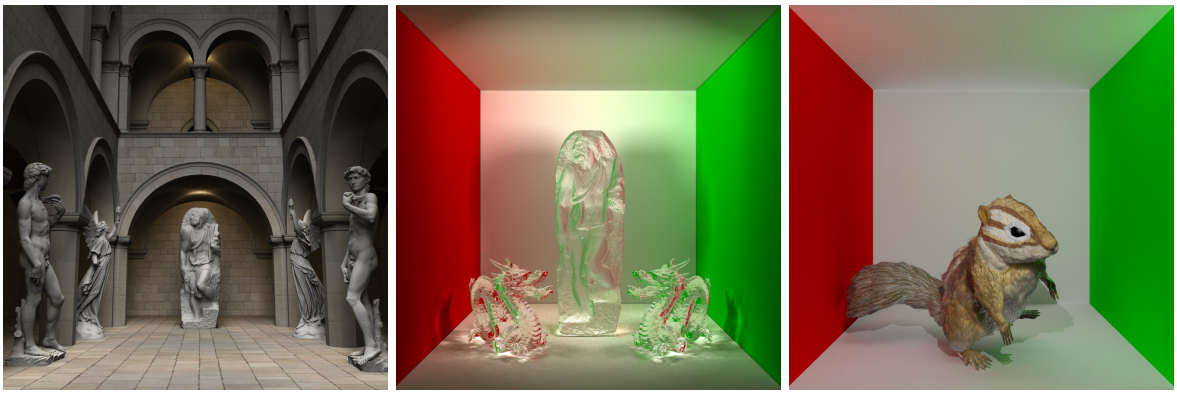


Figure 3.5: The left image shows the result of our method applied to path tracing of a Sponza model with a St. Matthew model, two Lucy, and two David models. This Sponza scene consists of 104 million triangles, requiring 12.8 GB for the original meshes and their acceleration hierarchies. The middle and right images show photon mapping results of a transparent St. Matthew model consisting of 128 M triangles in the Cornell box with two transparent dragon models, and a furry squirrel modeled with 32 million hair strands in the Cornell box. The St. Matthew and squirrel scenes take 15.7 GB and 8.2 GB respectively. These two global illumination methods generate many incoherent rays to render these images. By reordering such rays, we achieve more than one order of magnitude performance improvement in a machine with 4 GB main memory, compared to without reordering rays. This performance improvement is caused by the improved ray coherence.

main memory space to an *in-core* ray buffer. Once the in-core buffer is full, we push these rays into an *out-of-core* ray buffer on the disk and then store the next rays in the in-core ray buffer. We do not pose any restriction on the size of the out-of-core ray buffer. If there are no more rays that we can generate, we sort the rays stored in the in-core and out-of-core ray buffers.

We test our method with two global illumination methods: path tracing and photon mapping, both of which generate many incoherent rays to produce realistic visual effects. We generate primary rays in Z-curves for all the tests.

3.3.1 Path Tracing

The left image of Fig. 3.5 shows an unbiased rendering image of the St. Matthew, two Lucy, and two David models in the Sponza scene using a path tracing method [65, 53]. This scene consists of 104 M triangles; we do not use any instancing for duplicate models. BVHs and meshes of models in the scene take 12.8 GB. Since our ray tracer with the 32 bit test machine can use only 3.05 GB, main memory of the machine can cache 23.8% of all the data for our out-of-core ray tracer. To illuminate the scene, we use 8 area lights. We generate 100 primary rays (i.e., paths) per pixel and use simple importance sampling by generating shadow rays to the lights. In this configuration, we generate 361 M rays at each frame; 309 M and 26 M rays among all the generated rays are shadow and secondary rays respectively. We use the Russian roulette method to determine the path length.

In this scene, our method achieves a 16.83 times performance improvement over rendering without reordering rays. We also measure the number of the disk I/O accesses occurring during the access of meshes and BVHs (Fig. 3.6), by using the Windows built-in performance monitor tool, *perfmon*. By reordering rays, we reduce the number of the disk I/O accesses that occurred without reordering rays

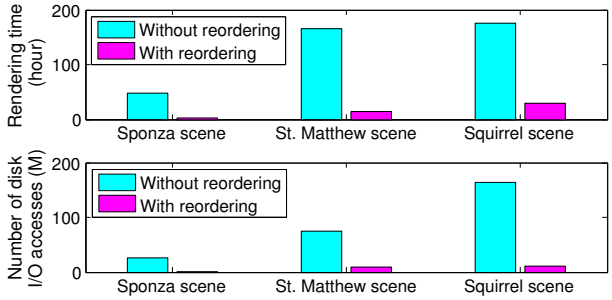


Figure 3.6: These figures show the overall rendering time and the number of the disk I/O accesses that occurred during rendering of the Sponza, the St. Matthew, and the squirrel scenes.

In-core ray buffer size	22MB	44MB	88MB	176MB
Rendering time (sec.)	10,541	10,460	10,314	10,459
Overhead (sec.)	1,039	879	754	693

Table 3.1: This table shows the overall rendering time and the total overhead of our method as a function of the in-core ray buffer size in the Sponza benchmark with 4 GB main memory.

by 93.6%. We also measure the average disk I/O access performance (MB/sec.) per disk I/O access. We found that reordering rays improves the disk I/O access performance by 208.5%. This is because the disk I/O accesses become more coherent and the disk can process these I/O accesses with a higher reading performance during the random accesses on BVHs and meshes. Because of these two factors, the reduction of disk I/O accesses and the improvement of disk I/O performance, we achieve more than an order of magnitude performance improvement when caching only 23.8% of all the data in main memory.

3.3.2 Photon Mapping

The middle image of Fig. 3.5 shows a rendering of the transparent St. Matthew and two transparent dragon models in the Cornell box scene using the photon mapping method [31]. This St. Matthew scene consists of 128 M triangles and takes 15.7 GB for its meshes and BVHs; therefore, the machine can cache only 19% of the total model size. We use 4 area lights, generate 25 primary rays per pixel and 10 final gathering rays, and use 100 samples for the irradiance estimation; 26 M shadow and 91 M secondary rays are generated among all the generated 124 M rays at each frame. In this configuration, our method achieves a 12.28 times improvement compared to rendering without reordering rays. By reordering rays, we reduce 88% of the disk I/O accesses and improve the disk I/O performance by 141%.

We also test a furry squirrel model that has 32 M hair strands. Each hair strand is represented as 8 cylinders. This model, shown in the right image of Fig. 3.5, consists of 256 M cylinders and takes 8.2 GB for its cylinders and BVHs. This model has lots of small hairs and thus is considered as a difficult benchmark for computing a high-quality simplification. Moreover, since there are a lot of complex occlusion among furs, our approximation method for hit points using simplified representations may not work well in this squirrel scene. Also, we have to recursively generate many secondary rays until the accumulated opacity is higher than a threshold (e.g., 0.9), because of the semi-transparent property of furs. We use a single point light, generate 25 primary rays per pixel, and 10 final gathering with 300 samples for the irradiance estimation; 8 M shadow and 67 M secondary rays are generated among all the generated 81 M rays.

Complexity of simplified model	0.0125%	0.05%	2%	8%
Rendering time (sec.)	10,644	10,342	10,314	13,789
Overhead (sec.)	637	648	754	964

Table 3.2: This table shows the overall rendering time and the total overhead of our method as a function of model complexity of simplified models, represented in the percentage of the original model complexity, in the Sponza benchmark.

In this configuration, our method that uses HPH achieves 3.77 times performance improvement and reduces 85% of the disk I/O accesses over rendering without reordering rays. Note that reordering rays based on HPH shows a relatively low performance improvement in this scene, compared to other scenes. Although there are small differences among the approximate hit points computed from the simplified fur model, there can be big differences among the exact hit points, lowering the ray coherence in the sorted rays. However, we found that considering ray origins or directions in addition to approximate hit points can further improve the performance. For example, when we consider approximate hit points as well as ray directions within our ray reordering method, it achieves a higher improvement, 5.9 times improvement, over rendering without reordering rays.

3.3.3 Analysis

We discuss various factors that affect the performance of our method with the path tracing benchmark of the Sponza scene in this section, unless mentioned otherwise.

Performance vs. complexity of simplified models: The complexity of simplified models can affect the performance improvement of our ray reordering method. We measure the performance improvement caused by our reordering method with different complexities of simplified models (Table 3.2). We achieve the highest performance when we use simplified models whose model complexities are 2% of original models. Moreover, we also found that the performance of our method does not decrease much as we use drastically simplified models (e.g., 0.0125% of the original models for the simplified models).

Overhead: We also measure the total overhead of our method, which consists of computing approximate hit points and sorting rays stored in the ray buffer. We found that the total overhead of our method is 7% of the total rendering time when we use 2% of the original model complexity for the simplified models; sorting rays takes 65% of the total overhead. Also, about 55% of the total rendering time with reordering rays is spent on reading data from the disk, compared to 98% of the total rendering time measured without reordering rays.

Performance vs. ray buffer size: The performance improvement can be affected by the size of the in-core ray buffer. We measure the rendering time as a function of the size of the in-core ray buffer (Table 3.1). As the size is increased, we found that the overhead of our method is decreased. However, as we allocate more memory space for the in-core ray buffer, less memory space is used for other data such as meshes and BVHs. Therefore, we achieve the highest performance when we allocate 88 MB for the in-core ray buffer. However, the performance variation is rather minor in the tested range of the buffer size.

Cache-oblivious nature of our method: Our method uses Z-curves for reordering rays and has the cache-oblivious property caused by using the space-filling curve [85] that works with different cache parameters. Therefore, it can reduce cache misses occurring between different memory levels including

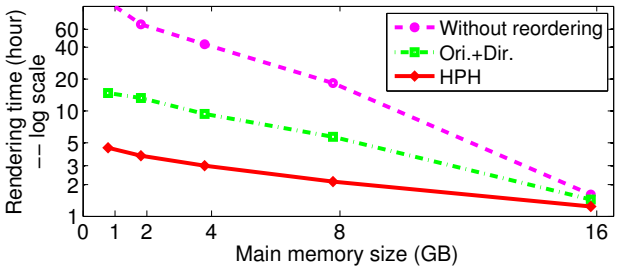


Figure 3.7: This graph shows the rendering time of path tracing in the Sponza scene with different physical main memory sizes, when we use our method or not. We also show the rendering time measured with a reordering method that considers ray origins as well as ray directions together (**Ori.+Dir.**).

L1/L2 caches, main memory, and disk. To demonstrate the cache-oblivious property of our method, we test our method with photon mapping of the Armadillo model consisting of 346 K triangles in the Cornell box (Fig. 3.2). The whole data of this small scene takes 43.5 MB, which fits into main memory. In this scene, we reorder rays when our in-core ray buffer is full, instead of dumping rays stored in the ray buffer to the out-of-core ray buffer. In this case, our method shows a 21% overall performance improvement by reordering rays. This improvement is caused by the improved ray processing throughput, although our method has an overhead of computing approximate hit points and sorting rays, which take about 14% of the overall rendering time. The ray processing throughputs is improved from 135 K rays per second (RPS) to 164 K RPS. We also measure the L2 cache miss ratios by simulating the 6MB wide 24-way set-associative L2 cache of our test machine. We observe more than two times cache miss reduction by reordering rays compared to without reordering rays.

Performance vs. cache size: The performance improvement of our method depends on how much portion of the data of a scene can be stored by different caches. To shed light on this factor, we measure the overall rendering time, as a function of the available memory size with and without using our ray reordering method (Fig 3.7). For this test, we use a 64 bit machine; note that the OS uses 0.2 GB space from the physical main memory. When we use 16 GB main memory, the whole data of the scene can be uploaded into main memory. Even in this case, our method improves the performance by 31% over rendering without reordering rays, because our method improves the cache utilizations of L1/L2 caches. As we decrease the memory size, the performance of ray tracing also decreases. Nonetheless, the performance with our ray reordering method decreases more gracefully. When caching 1.8 GB, 14.1% of the whole data, in main memory, our method shows a 17.8 times improvement. Even when the available memory size is 0.8 GB, 6.2% of the whole data, our method can render the Sponza scene without I/O thrashing. Also, as we reduce the memory size, the performance improvement of our method increases, since data access time takes a larger portion in the whole rendering time, which gives more room for improvements to our method.

Performance vs. layout: We have used cache-efficient layouts for meshes and hierarchies in this paper, to maximize the benefits of our ray reordering method. Also, the depth-first layout of a BVH has been also widely used in many ray tracers [87].

We also measure the performance of our ray tracer with the depth-first layout, to see how much performance degradation our method can have. Even if we use the depth-first layout, we observe only 14% performance degradation over using the cache-efficient layout.

Multi-core architectures: We also test our method in the 32 bit machine with a quad-core CPU.

Scene	**HPH**	**Ori.**	**Ori.+Dir.**	**HPH+Ori.**	**HPH+Dir.**	**Pharr97**
Sponza scene	16.65	5.16	5.19	7.97	6.36	13.69
St. Matthew scene	12.28	1.7	2.29	4.03	5.36	28.61
Squirrel scene	3.77	0.89	3.49	4.35	5.91	N/A
Small scene	1.21	0.97	1.23	1.18	1.17	N/A

Table 3.3: This table shows performance improvements (times) of tested sorting measures over the overall rendering time without reordering rays in our benchmark scenes. **Ori.**, **Ori.+Dir.**, **HPH+Ori.**, **HPH+Dir.**, and **Pharr97** represent sorting rays based on ray origins, ray origins combined with ray directions, hit points combined with ray origins, hit points with ray directions, and using the cache-aware method of Pharr et al. [1997] respectively.

Our reordering method can be easily parallelized since ordering keys of hit points is easily computed by a few simple bit operations. Also, the 2-way merge sort method that we used for sorting rays is easily parallelized. We measure the performance improvement by reordering rays when we use four threads for ray tracing and our reordering method. By reordering rays, we achieve 10.5 times improvement over without reordering rays when we use four threads in the quad-core CPU machine.

3.4 Comparisons

We compare the performance of our method (**HPH**) with those of other reordering methods that include a seminal ray reordering method (**Pharr97**) proposed by Pharr et al. [52] and simple ray reordering methods that sort rays based on ray origins (**Ori.**) and ray origins with ray directions (**Ori.+Dir.**). We also test two variations of our method that sort rays based on hit points with ray origins (**HPH+Ori.**) and hit points with ray directions (**HPH+Dir.**). Note that **Pharr97** is a cache-aware method, while all the other methods including ours are cache-oblivious.

All the techniques except for **Pharr97** are implemented within our space-filling curve based reordering framework described in Sec. 3.2.3; we use 5 dimensional grids with $k = 12$ for **Ori.+Dir.** and **HPH+Dir.**, and use 6 dimensional grids with $k = 10$ for **HPH+Ori.**. For **Pharr97**, we divide the scene into to a set of chunks, each of which takes about 32 MB and has its own ray queue. We also construct a higher level kd-tree whose leaf node contains a single chunk. We choose to use the kd-tree for the higher level hierarchy instead of a BVH, since the kd-tree can provide early terminations of rays. Then, we construct a low-level BVH for each chunk, to perform a fair comparison with our method that uses BVHs. We follow the scheduling method for chunks as proposed by Pharr et al. [52].

Reordering methods based on HPH show higher performance improvements over simple cache-oblivious ray reordering methods based on ray origins or ray directions (see Table 3.3). **HPH** shows higher performances over **Ori.** and **Ori.+Dir.** in all the tested scenes, except for the small Armadillo scene (Fig. 3.2). Even in the small scene, **HPH** has a shorter traversal time than **Ori.+Dir.**. However, the overhead of **HPH**, especially computing approximate hit points, is higher than that of **Ori.+Dir.**. As a result, **HPH** shows a slightly lower performance than **Ori.+Dir.**. **HPH** also shows higher performances than other sorting methods that consider **HPH** as well as **Ori.** (or **Dir.**) in all the tested scenes, except for the squirrel scene that has complex occlusions.

Our method shows about two times slower performance than **Pharr97** in the St. Matthew scene. However, our method shows a slightly higher (e.g., 23%) performance improvement than **Pharr97** in the

Sponza scene. **Pharr97** like most previous cache-aware methods [47, 6] reorders rays as they traverse their scenes or acceleration hierarchies, because the data access patterns of rays are known during the scene or hierarchy traversal. The main benefit of these methods is that since the data access patterns of rays to the hierarchies and meshes are known during the traversal, sorting rays with this information can result in a low number of cache misses and even higher performances than our method, as demonstrated in the St. Matthew scene. However, this approach requires a tight integration between the ray reordering module and the ray processing module, causing a complication to the overall ray tracing system and a major restructuring of existing systems in order to use these reordering methods. Although our method shows a lower performance over **Pharr97** in the St. Matthew scene, our method did not show a drastically lower performance and shows even higher performance in the Sponza scene. Therefore, we argue that our method can be useful and widly applied to many existing ray tracing system, since it is simple to implement, highly modular, and cache-oblivious in addition to showing comparable performances with **Pharr97**.

Chapter 4. Robust Image Denoising using a Virtual Flash Image

4.1 Image Denoising using a Virtual Flash Image

Our goal is to denoise images generated by various Monte Carlo ray tracing algorithms. Since rendered images can have image features buried under noise, naïve applications of filtering techniques proposed in the image processing field may not provide satisfactory results. For example, Xu and Pattanaik [83] pointed out that a naïve usage of bilateral filtering to noisy rendered images works poorly. In order to address this issue, we propose to use a *virtual flash image* that serves as an estimator of image features in the input noisy image. Examples of virtual flash images are shown in Fig. 4.7.

4.1.1 Generating Virtual Flash Images

We aim to discern all the image features from noise in the input image without taking additional ray samples. To achieve this goal, a virtual flash image is constructed by summing two different components: 1) a part of the illumination from the original light sources, and 2) additional illumination from a virtual flash point light located at the viewing position.

To create the virtual flash image, we reuse subsets of light paths (and their shading results) that are generated by the original rendering method with the original light sources. Among the light paths of the original light sources, we keep all the ray paths that interact with specular and glossy materials to compute the virtual flash image. On the other hand, for ray paths interacting with diffuse materials, we keep only the ray paths that have at most one diffuse bounce from the viewpoint (e.g., direct illumination). This is because shading for multiple diffuse bounces causes significant noise in the computed illumination values. This limited set of ray paths can capture most high-frequency image features (e.g., textures, shadow boundaries with or without reflections and refractions) in the virtual flash image. However, since we ignore some ray paths, some features (e.g., caustics in Fig. 4.8-(a)) cannot be captured in the virtual flash image. We explain how to robustly detect such missing features during our denoising process in a later section.

The additional virtual flash light source is crucial for capturing image features in regions where the original lights do not cast direct illumination, as shown in Fig. 4.8. The conventional method for approximating indirect illumination uses an approximate ambient term [55], but this approach can only capture a limited set of high-frequency features (e.g., textures) introduced by primary rays. The virtual flash light ensures the virtual flash images capture a larger variety of high-frequency features (e.g., edges on specular surfaces) through existing shading codes compared to the conventional method (Fig. 4.8). Additionally, the virtual flash images with considering the additional virtual flash light can be constructed quickly because they do not require any additional ray samples. Specifically, we do not perform any visibility tests nor create shadows from the virtual flash light, to avoid adding shadows that do not exist in the input image.

We set the intensity of the virtual flash light such that it can compensate the loss of indirect illumination in the virtual flash image; this can be done quite efficiently by comparing the original image

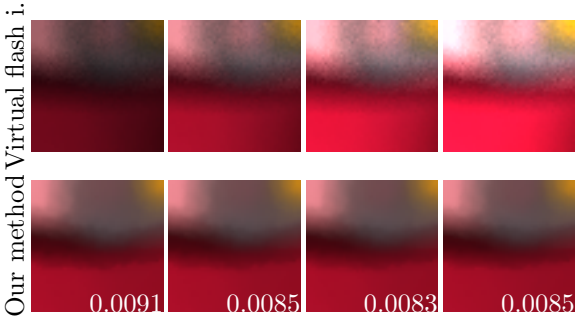


Figure 4.1: The first row shows virtual flash images, as we change the intensity of the virtual flash light from 2 to 6 times over the intensity of the original light in the toaster scene. The denoised results with different virtual flash images are shown in the second row. The virtual flash light intensity, 4.7 times to the original light, generated by our method produces the lowest RMS error.

Figure 4.2: Our denoising results w/ and w/o considering the virtual flash image within our non-local means filtering. Considering the virtual flash image, we can preserve fine details that are captured in the virtual flash image. A-Trous filter [13], which is based on geometric information, fails to preserve both refracted and reflected edges on the glass. The input and reference image are generated with 64 and 2,500 ray samples per pixel.

and the virtual flash image rendered without the virtual flash light. We found that this simple heuristic works very well. Furthermore, we found that a wide range of values that deviate from the intensity computed by the heuristic also work well, and robustly produce almost identical denoising results, as shown in Fig. 4.1.

Existing image-based denoising methods [45, 13, 3, 63] use geometric features (e.g., normals and depths stored in the G-buffer) as edge-stopping functions. Albedo (e.g., colors) can also be considered for preserving texture edges.

Unfortunately, reflected and refracted edges of transparent objects (shown in Fig. 4.2) cannot be easily preserved because they are not simply created by the geometries and albedo of transparent objects.

In addition, the correct estimation of the geometric information of blurred regions generated by defocus or motion blur effects is not trivial; averaging depths and normals across edges may give incorrect geometric values as pointed out by Bauszat et al. [3].

The key concept of using virtual flash images is that, although limited, we combine geometric features and albedo of surfaces intersected by multiple rays into a single image through actual shading. This seemingly simple concept adds a powerful capability of capturing discontinuities with non-diffuse objects such as a glass shown in Fig. 4.7.

4.1.2 Denoising using a Virtual Flash Image

Given a Monte Carlo ray tracing method that generates an input noisy image N, we denoise the input image to obtain an output image D. Using a virtual flash image F and non-local means filtering [5] (i.e., a patch-based bilateral filtering), the output D_p of pixel p is computed as follows:

$$D_p = \frac{1}{k(p)} \sum_{p' \in H(\Omega_p)} g_s(|p' - p|) g_r\left(|u(F_{p'}) - u(F_p)|\right) N_{p'}, \tag{4.1}$$

where $k(p)$ is a normalization term, the function $u(F_p)$ denotes a patch of pixels around p in the virtual flash image, and Ω_p is defined as the pixels in a square denoising window centered on pixel p. g_s and g_r are spatial and range Gaussian filters that have standard deviations of σ_s and σ_r, respectively. These two filters serve as edge-stopping functions; the range filter computes weights based on the intensity difference of pixels, whereas the spatial filter sets weights based on the spatial distance of pixels. Note that our range filter computes weights using the virtual flash image F, which has much reduced noise compared to the input image N.

The patch of a pixel p, $u(F_p)$, in our denoising framework (Eq. 4.1) is defined as an m by m window whose center is located at pixel p. In order to compute the distance between the two patches $u(F_p)$ and $u(F_{p'})$ in the range filter, we compute a weighted Euclidean distance among all the corresponding pixels in the two patches, as suggested by Buades et al. [5].

By considering a virtual flash image within our denoising framework, we can preserve most image features during the denoising process; denoising results with and without considering our virtual flash image are demonstrated in Fig. 4.2. However, there are some other features (e.g., caustics, color bleeding) that are not captured well by the virtual flash image. Such features can be undesirably removed, especially when we use a large denoising window. It is, however, a challenging problem to set the optimal denoising window size that simultaneously preserves image features and reduces noise [83]. Therefore, existing image-based denoising methods simply leave the user to find such a window size by trial and error, which can easily lead to over-blurred or under-smoothed images [34]. Instead of Ω_p, we propose to use $H(\Omega_p)$, a set of homogeneous pixels, for preserving those edges (e.g., caustics) that the virtual flash image does not include.

4.1.3 Robust Denoising with Homogeneous Pixels

Instead of seeking the optimal denoising window size, we propose a simple, yet effective, approach to suppress excessive blurring. Our approach is to identify *homogeneous pixels*, pixels that are considered statistically equivalent based on confidence intervals of the true means of pixels.

Given a pixel p, we define n_p, $\bar{x}_p$, and s_p, to indicate the ray sample count for the pixel, the sample mean, and the standard deviation computed with the observed ray samples for that pixel, respectively. We use μ_p to indicate the true mean of samples for the pixel p. We adopt a large denoising window with a large spatial Gaussian in order to increase the probability of identifying pixels that are correlated with p, and thus achieving better denoising quality. The key is to use only homogeneous pixels ($H(\Omega_p)$ in Eq. 4.1), from all of those in Ω_p to smooth out pixel p.

We define homogeneous pixels $H(\Omega_p)$ as a set of pixels whose sample means are within a confidence interval of the unknown true mean μ_p. We use the t-distribution to construct the confidence interval for the unknown true mean μ_p with a confidence level of $1 - \alpha$ [29].

As a result, the homogeneous pixels $H(\Omega_p)$ of a given pixel p are defined as:

$$\left\{ p' | p' \in \Omega_p, \bar{x}_p - \frac{t_{\alpha/2,n_p-1} s_p}{\sqrt{n_p}} \leq \bar{x}_{p'} \leq \bar{x}_p + \frac{t_{\alpha/2,n_p-1} s_p}{\sqrt{n_p}} \right\}. \tag{4.2}$$

In the above equation, $t_{\alpha/2,n-1}$ is a *critical point* in a two-sided t-interval, which is defined as $P(X \geq t_{\alpha/2,n-1}) = \alpha/2$ where X is a random variable drawn from the t-distribution with $n-1$ degrees of freedom. In practice, the use of a pre-computed lookup table for the critical point is recommended for efficient computation [29]. The size of the table is negligible, as the definition of the critical point only depends on the number of samples and the user-specified value α.

This definition of homogeneous pixels is closely related to testing the null hypothesis $H_0 : \mu_p = \mu_{p0}$ against the alternative hypothesis $H_A : \mu_p \neq \mu_{p0}$, where μ_{p0} denotes a specified value that we want to test. For the specified value, we use the neighboring sample mean $\bar{x}_{p'}$. The null hypothesis is then accepted if $|t| \leq t_{\alpha/2,n_p-1}$, where the test statistic $t = (\bar{x}_p - \bar{x}_{p'})/(s_p/\sqrt{n_p})$. The acceptance of the null hypothesis indicates that the tested value $\bar{x}_{p'}$ (i.e., μ_{p0}) is a plausible value of the unknown mean μ_p.

We found that the commonly adopted confidence level of 99% (i.e., $1-\alpha = 0.99$) works very well in practice. An example of homogeneous pixels and their weights is shown in Fig. 4.3. Note that the virtual flash image does not have any information about the caustics; this example is a zoomed-in inset of the leftmost caustics shown in the wedding-band scene (the last row of Fig. 4.10). Owing to homogeneous pixels, our method successfully preserves such features.

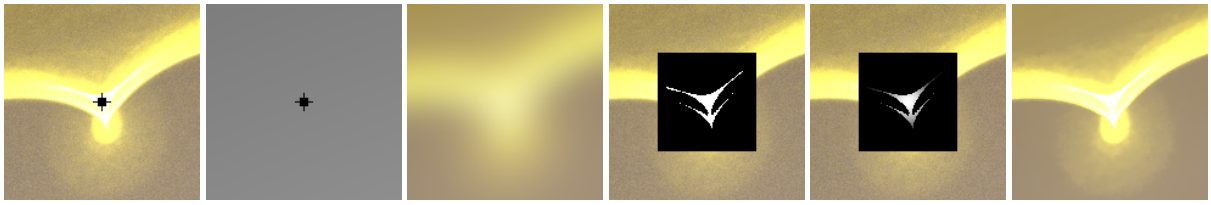
To define the homogeneous pixels, we assume that a sequence of independent identically distributed (i.e., iid) random samples for pixel p is generated from Monte Carlo ray tracing. We compute a sample mean $\bar{x}_p$ of all the samples for the pixel p. The central limit theorem [29] indicates that the distribution of sample means is closely approximated by a normal distribution, regardless of the actual distribution of individual ray samples. The accuracy of the approximation improves as the number of ray samples per pixel increases. The confidence intervals of the unknown true means are also approximate. However, empirical studies [73, 25] show that confidence intervals in Monte Carlo ray tracing are reasonably well approximated by a relatively small number of ray samples (e.g., $n_p = 20$). We also found that the coverage probability of the approximate confidence intervals (Table 4.1) becomes very close to the confidence level as the number of ray samples increases. As a result, we also assume that the true mean μ_p for a pixel p is in the interval $(\bar{x}_p - \frac{t_{\alpha/2,n_p-1} s_p}{\sqrt{n_p}}, \bar{x}_p + \frac{t_{\alpha/2,n_p-1} s_p}{\sqrt{n_p}})$ with a probability of $1-\alpha$.

The concept of homogeneous pixels is not entirely novel, because it is based on well-known statistics (e.g., normal theory). For example, the anisotropic method [45] assumes that the sample mean in each pixel is a normally distributed random variable in a color space, and the distance between distributions in adjacent pixels is defined in a statistical manner. In contrast, however, our method uses only homogeneous pixels, rather than all the neighboring pixels, for denoising. Kervrann and Boulanger [34] proposed the use of confidence intervals in image processing for selecting neighboring, statistically equivalent, patches. Nonetheless, in the next section we present new stochastic error bounds using sampling information obtained through Monte Carlo ray tracing.

4.1.4 Stochastic Error Bounds

One way of defining the quality of any denoising method is to measure the difference between the denoised value D_p and its reference value μ_p that is generated by a Monte Carlo ray tracing method at a pixel p. In particular we measure the positive distance, i.e. denoising error, between those two values, $|D_p - \mu_p|$.

The assumption to define homogeneous pixels is that given iid random samples generated by MC



(a) Input noisy i. (b) Virtual flash i. (c) Ours with Ω_p (d) $H(\Omega_p)$ (e) Weight (f) Ours with $H(\Omega_p)$

Figure 4.3: Homogeneous pixels $H(\Omega_p)$ (white pixels) given a pixel p centered at the denoising window (black square box), and their weights in high-frequency caustics of the wedding-band scene (the last row of Fig. 4.10). The images (c) and (f) are denoised results without and with only considering homogeneous pixels respectively within our method. The virtual flash image (b) does not contain any information for caustics, and thus the caustics are blurred without considering homogeneous pixels (c).

Ray Samples	4	16	32	64	128
$1-\alpha = 0.95$	0.855	0.901	0.921	0.938	0.942
$1-\alpha = 0.98$	0.899	0.929	0.945	0.960	0.964
$1-\alpha = 0.99$	0.921	0.943	0.956	0.968	0.972

Table 4.1: Coverage probability of confidence intervals for unknown means μ_p in the toaster benchmark. This result shows a slightly underestimated coverage compared to a confidence level of $1-\alpha$, especially with low ray samples (e.g., 4), but the coverage probability becomes very close to the confidence level afterwards.

ray tracing, the unknown true mean is within its confidence interval with a probability. The error of the original image rendered by unbiased Monte Carlo ray tracing reduces in the order of $n_p^{-1/2}$, but a denoising method can introduce a systematic error (i.e., bias). For example, a denoising method with a fixed parameter is not consistent, as demonstrated by Sen and Darabi [63]. We show that our denoising method considering only homogeneous pixels is consistent, and its stochastic error bound reduces in the order of $n_p^{-1/2}$.

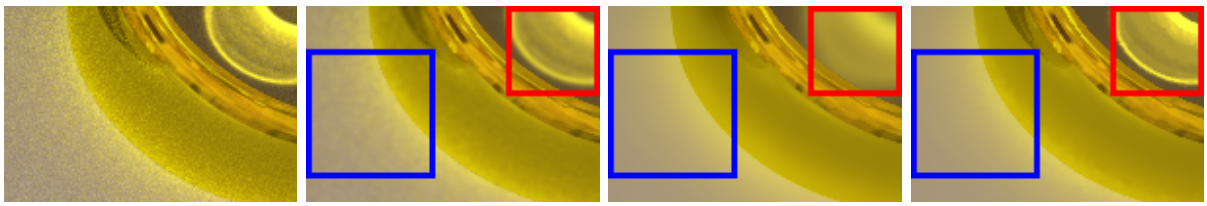
Theorem 4.1.1 *The denoising error of our method given a pixel p is stochastically bounded with a probability that is greater than or equal to* $1-\alpha$ *as the following:*

$$P\left(|D_p - \mu_p| \leq 2t_{\alpha/2,n_p-1}s_p{n_p}^{-1/2}\right) \geq 1-\alpha.$$

We prove this theorem by taking advantage of our definition of the homogeneous pixels. The detailed proof for the theorem is in Appendix 4.4.

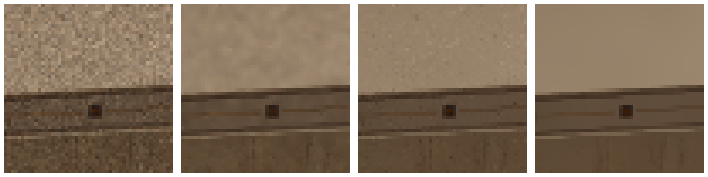
According to the theorem, the error introduced by our denoising method is stochastically bounded, and its stochastic bound reduces as the order of $n_p^{-1/2}$, as we increase the number of ray sample n_p for the pixel p.

Note that this property is satisfied irrespective of denoising parameters (e.g., denoising window size) used in our method. This is a notable advantage compared to prior work, as this property lessens the difficulty of choosing denoising parameters and makes our method more practical. Furthermore, it allows our denoising method to have a large denoising window, which increases the probability of finding similar pixels without introducing blurring artifacts as illustrated in Fig. 4.4.



(a) Input noisy image (b) Denoising with Ω_p, $\sigma_s = 1$ (c) Denoising with Ω_p, $\sigma_s = 5$ (d) Denoising with $H(\Omega_p)$, $\sigma_s = 5$

Figure 4.4: Denoising results with a small denoising window size (b), a large one (c), and homogeneous pixels (d). The image (b) preserves high-frequency caustics (shown in the red box), but shows low-frequency noise on the floor (shown in the blue box). On the other hand, the image (c) shows smooth results on the floor, but blurs the caustics. Our method considering homogeneous pixels shows smooth results on the floor, and preserves high-frequency caustics even though we use the large denoising window.



(a) Input (b) 1st-step (c) 2nd-step (d) Both steps

Figure 4.5: Denoised images performed with the first-step only, the second-step only, and both steps of our two-step denoising process. The input image is generated with 64 ray samples per pixel.

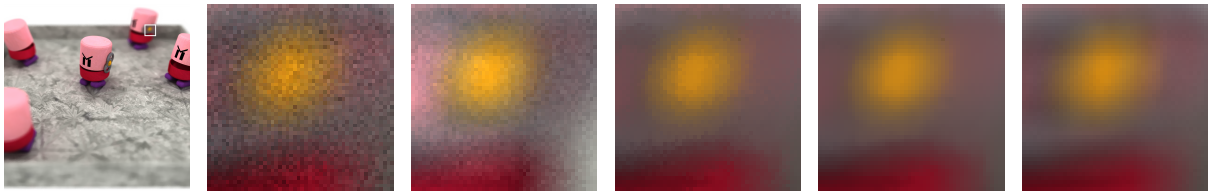
4.1.5 Two-Step Denoising Process

In order to robustly denoise input noisy images, we use *two-step denoising process*, which performs our denoising two times. In the first step, we perform our denoising with a small denoising window size (e.g., 7 by 7) and the 99.8% confidence level, mainly for reducing the variance of the input noisy image. In the second step, we re-apply our denoising with a big denoising window size (e.g., 31 by 31) and the 99% confidence level.

If we perform the second step alone, the denoised image has a small bias. However, we found that some pixels are not smoothed out in the denoised images (Fig. 4.5), because of the short confidence interval. This freckle-like result happens when input images have a high level of noise with low ray samples. More specifically, this occurs when two pixels have different sample means with small variances, even though they are supposed to have similar distributions of ray samples. If we relax the confidence interval (i.e., wide confidence interval), we can avoid this freckle-like results in the denoised image, but introduce a bigger bias.

4.2 Results

We have implemented two Monte Carlo ray tracing methods, photon mapping and path tracing in a CPU-based ray tracer, and applied our method to images generated by these two methods. We use 1280 by 960 image resolutions. We do not perform anti-aliasing for all the images shown in this paper, to



(a) Ours (b) Inset (c) Virtual flash i. (d) Without patch (e) With patch (f) Reference

Figure 4.6: An inset (b) of an input noisy image (generated with 64 ray samples) in the toaster scene, with its virtual flash image (c). Denoised results with and without patch-wise weight computation are shown in (e) and (d), respectively. Note that the denoised result with non-local means filtering is very similar to the reference image (f) generated with 10,000 ray samples per pixel.

highlight noise contained in images; our method can be naturally combined with anti-aliasing and shows better results by using it together. We perform various tests on a PC with Intel Core i7 at 3.3 GHz and 4GB of memory. We have also implemented a GPU version of our denoising method on an NVIDIA GeForce GTX 580.

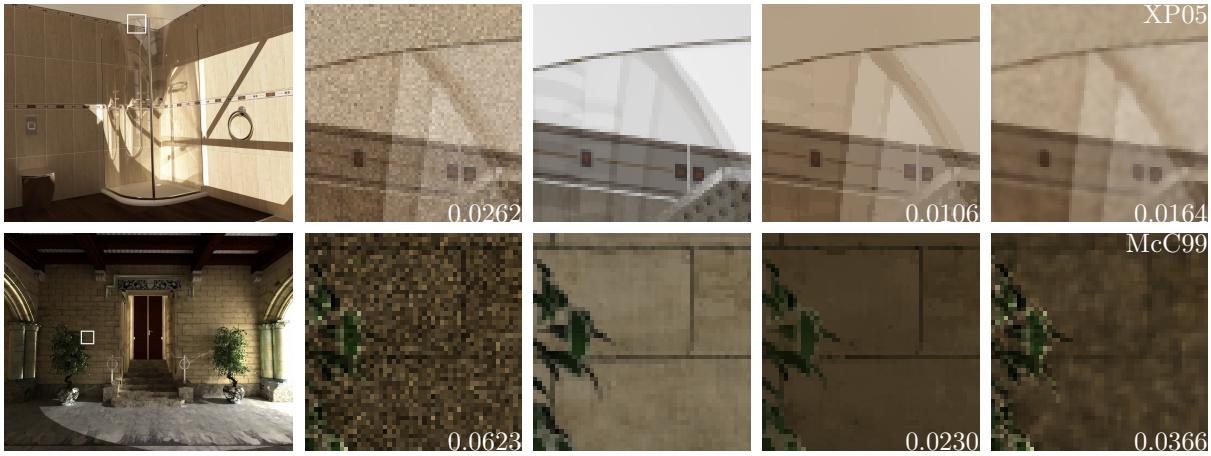
For all the tests, the denoising window size for Ω_p is set as 31 by 31 pixels, and σ_s of the spatial filter used in Eq. 4.1 is set as one third of the width (and height) of the denoising window size. We adjust σ_r for the range filter, to be linearly proportional to the standard deviation of the mean of ray samples considered in the virtual flash image, as it has been known that σ_r should be chosen depending on the noise level of an input image to achieve a high denoising quality [88].

Benchmarks: We have tested five benchmark scenes that have different characteristics: 1) outdoor, 2) bathroom, 3) shower booth, 4) toaster, 5) wedding-band benchmarks. The outdoor scene is shown in Fig. 4.14, and other scenes are shown in Fig. 4.10 from the top to the bottom in the order same to what they are mentioned here.

The outdoor scene (4.7 M triangles and 90 MB JPEG textures) has high geometric complexity on the two plants and high texture complexity on most parts of the scene. The bathroom (470 K triangles) has many specular objects such as mirrors and detailed textures on most parts of the scene. In both the bathroom and outdoor scenes, indirect illuminations are dominant. The shower booth scene (470 K triangles) contains glossy objects (e.g., metals and a trash can), and the scene has a glass window that causes strong caustics. The toaster scene (11 K triangles) is rendered with depth-of-field. The wedding-band scene shows high-frequency caustics. We compute the input images by path tracing for the outdoor and toaster scenes, whereas all the other images are generated by photon mapping.

Our denoising method robustly handles different materials such as diffuse, specular, and transparent materials (e.g., sinks of the bathroom, mirrors, and the shower booth). Our method also preserve complex geometric features (e.g., plants in the outdoor scene) and texture features (e.g., brick wall in the outdoor scene), whereas existing methods tend to fail (Fig. 4.10). The denoised images by our method preserve caustics in the shower booth and the wedding-band scenes. Furthermore, our method can handle other complex illumination effects such as the depth-of-field in the toaster scene. As shown in the third row of Fig. 4.10, the depth-of-field effect causes noise in the virtual flash image, but the level of noise in the virtual flash image is much reduced compared to that of the input image, as it ignores diffuse interreflections. As a result, denoising with the virtual flash image brings higher improvement out than denoising with the original noisy image.

RMS comparisons: We measure the RMS error, $(\frac{1}{|D|}\sum_{p\in D}|D_p-\mu_p|^2)^{1/2}$, between the reference and



(a) Our method (b) Input image (c) Virtual flash image (d) Our method (e) Previous methods

Figure 4.7: Results of our denoising method with 64 ray samples per pixel. Our method takes input images (column (b)) and significantly reduces noise level while keeping salient image features (columns (a) and (d)). The key idea is the use of virtual flash images (column (c)), which capture various image features without additional samples. Our method achieves not only visually better results, but also numerically more accurate results than the prior image-space denoising methods (column (e)). The numbers at the lower right corners the root mean square (RMS) errors computed from the reference solutions.

denoised images, where $|D|$ is the number of pixels of the denoised image D. We also measure the RMS error between the reference and input noisy images with different ray samples in the outdoor scene.

The bilateral filter proposed by Xu et al. [83] does not reduce the RMS error from 512 ray samples per pixel because it uses a fixed denoising parameter. However, the RMS error of our method continues to decrease (Fig. 4.9), as we have more ray samples. This result is achieved without changing any denoising parameters. The wavelet-based image denoising method [48] and anisotropic filtering [45] also reduce the RMS error, as we have more samples. This is because these methods control the level of denoising (i.e., blurring) according to variance of the sample mean of each pixel. Nonetheless, the RMS error of our denoised image shows the best results among all the tested methods from 4 to 2 K ray samples per pixel. This demonstrates that our denoising method can be robustly applied to rendered images that are generated with various numbers of samples without tweaking denoising parameters.

Computational overhead: To construct virtual flash images, we reuse ray sample information generated by the Monte Carlo ray tracing. Thus, creating virtual flash images takes a minor portion (usually less than 2%) of the original rendering time (e.g., 284 ms and 1.8 s in the outdoor scene with 4 and 16 samples per pixel while it takes 23 s and 92 s for creating input images respectively). Our denoising method has a time complexity of $O(|N||P|\sigma_s)$, where $|N|$ is the number of pixels in the input noisy image N and $|P|$ is the number of pixels in a patch. Given the denoising window size with the tested image resolution, we use a CPU implementation using the *OpenMP* [12] library that used 8 threads for the computation. Computation takes an average of 18.4 s for a 5 by 5 patch size; we found that a bigger patch size does not yield better results. Because our method can be easily parallelized on GPU, a GPU implementation of our denoising method takes only 1.5 s on average. This computation time for our denoising process is small compared to the time spent on rendering scenes; for example, it takes 65 s and 282 s to generate an input noisy image with only four samples per pixel for the bathroom and

(a) Input noisy image (b) Direct illumination (c) Direct illumination + ambient (d) Virtual flash image

Figure 4.8: An input noisy image (generated with 16 ray samples) for the shower booth scene, where indirect illuminations are dominant. The virtual flash image (d) captures more high-frequency features compared to images with the direct illumination (b) and the additional ambient term (c).

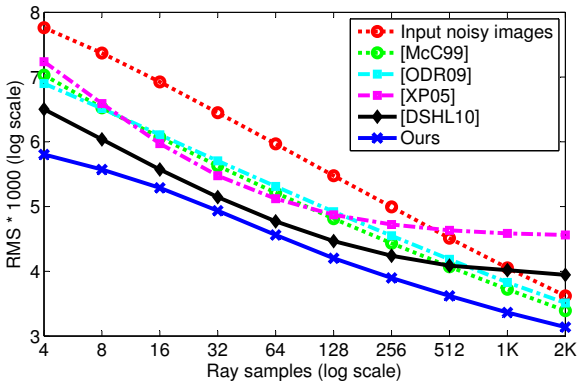


Figure 4.9: RMS comparisons of denoised results from 4 to 2 K ray samples per pixel over the reference image (generated with 32 K samples per pixel) in the outdoor scene.

shower booth scenes, respectively. We expect that our denoising process would be capable of performing interactively by adopting recent acceleration techniques for non-local means filtering [1]. Moreover, our CPU implementation can be further improved by up to a factor of four when vectorization for SIMD architectures is applied.

Equal-error comparisons: In the toaster scene, our method reduces the RMS error of the input image generated with 8 ray samples per pixel from 0.0714 to 0.0170. On the other hand, when we increase ray samples from 8 to 128 per pixel, the RMS error of Monte Carlo path tracing is reduced to 0.0182, a higher than the RMS error of the denoised image with 8 ray samples. In this case, our method achieves more than 16 times performance improvement given the same RMS error, compared to generating more ray samples. A similar improvement is achieved for the outdoor scene (Fig. 4.9). Our method spends additional 3.2 s to the time (8.4 s) taken to generate the input image with 8 ray samples per pixel, while generating 128 ray samples per pixel takes 128 s. As a result, our method achieves over 11 times improvement in terms of the wall-clock time given the same RMS error.

Denoising animations: Our denoising method can be easily extended to denoising animations to reduce temporal artifacts such as flickering. For denoising an animation, we treat a stack of animation frames as a 3D volumetric data, and simply extend the 2D square denoising window and patches into 3D cubic denoising window and patches for the 3D volumetric data. When iid random samples are generated by Monte Carlo ray tracing, the stochastic error bound of our method with homogeneous pixels is still valid even for animations. If underlying Monte Carlo ray tracing methods employ adaptive sampling

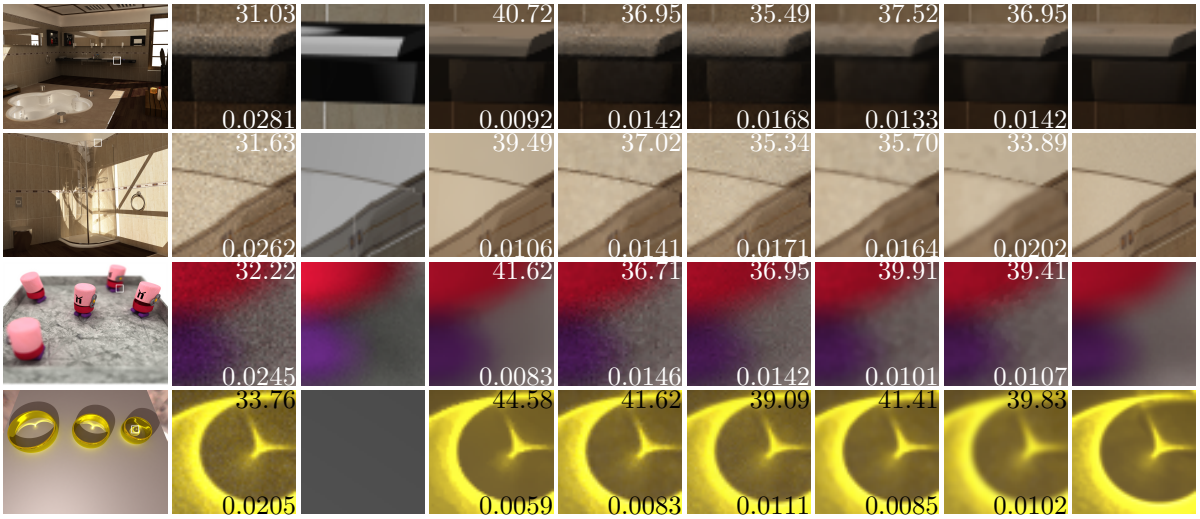


Figure 4.10: The column (a) shows denoised images by our method, followed by zoomed-in insets of the input noisy images (b), our virtual flash images (c), results of our method (d), anisotropic method [45] (e), wavelet-based denoising [48] (f), a variant of bilateral filtering [83] (g), A-Trous filtering [13] (h), and reference (i). The input noisy images are generated with 64 ray samples per pixel from the first to the third row, and with 16 ray samples per pixel in the forth row. The reference image in the second row are generated with 2,500 ray samples per pixel, and other reference images are generated with 10,000 ray samples per pixel. The numbers in the top and bottom of figures are RMS and PSNR, respectively.

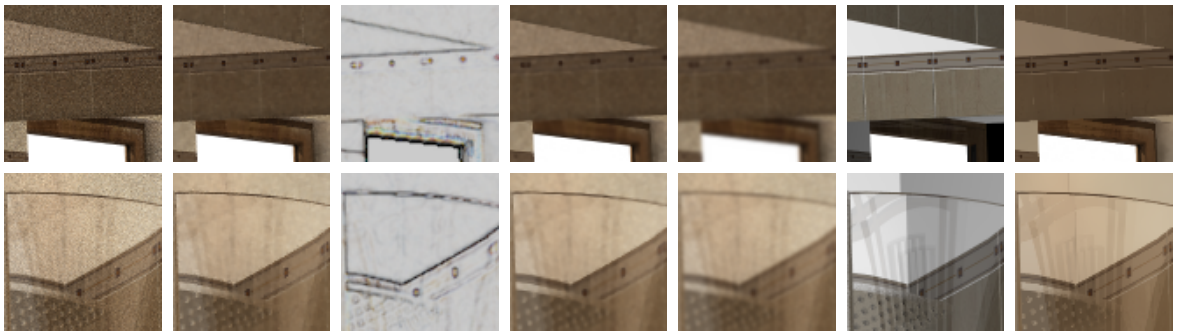
with considering correlations between frames, the stochastic error bound does not be maintained, because samples generated by adaptive sampling are not iid random samples.

Our denoising method takes linearly proportional time to the number of images that we need to consider for denoising a 2D image. In practice, just considering 5 images before and after an image for denoising works well, without leaving noticeable flickering artifacts; refer to the accompanying video for the results.

4.3 Comparisons

We have compared our method to existing image-space denoising methods including the anisotropic filtering [45], the bilateral filtering method proposed by Xu et al. [83], the wavelet-based image denoising method [48], and the geometry-aware A-Trous filter [13]. Our implementations for the prior methods follow the original guidelines shown in the original papers; we use all the edge-stopping functions (e.g., depth and color edge maps) proposed for anisotropic filtering. We have tested various settings and used a setting that works best across all the scenes. We have observed that finding proper denoising parameters for these image-space denoising methods is nontrivial except for ours and the wavelet-based method.

Fig. 4.10, 4.12 and 4.14 show comparison results; Fig. 4.10 and 4.14 show comparison results with low ray samples (e.g., 16 to 64), while Fig. 4.12 with high sample counts (e.g., 1 K). The anisotropic filtering method [45] computes various edge maps and attempts to preserve them; edge maps employed in this method are shown in Fig. 4.11. Nonetheless, as we perform iterations to filter out residual noise, those edges are affected more, leading to over-blurred images. The bilateral filtering method by Xu et al. [83] uses the blurred image as its range function, mainly to reduce high-frequency noise. As a result, their



(a) Input i. (b) [45] (c) Color edge map for (b) (d) [83] (e) Range function of (d) (f) Virtual f. (g) Ours

Figure 4.11: Denoising results and edge stopping functions used in the anisotropic method proposed by McCool [45], the bilateral method proposed by Xu et al. [83], and our method. The input noisy images (a) of the bathroom (the first row) and shower booth scene (the second row) are generated by photon mapping with 64 ray samples per pixel. Normal and depth maps used in the anisotropic method are omitted in this figure.

(a) Input noisy i. (b) Ours (c) [45] (d) [83] (e) [48] (f) Reference

Figure 4.12: Denoising results with 1K ray samples in a zoomed-in region of the outdoor scene. RMS errors are shown in images.

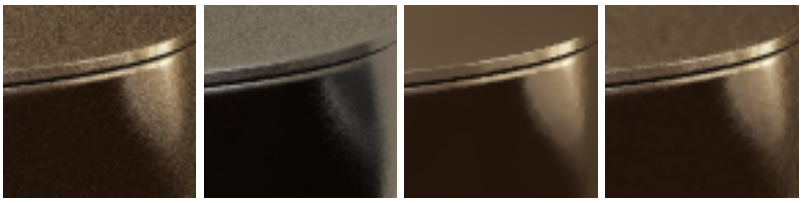
approach produces blurry denoised images, because the blurred image cannot have high-frequency edge information. The wavelet-based method [48] unfortunately suffers from ringing artifacts that wavelet-based methods commonly introduce. The A-Trous filter [13] can preserve the edges introduced by geometric discontinuities, but it fails to preserve the edges generated by complex illuminations such as refraction, reflection, caustics, and defocus effects. Overall, existing methods show excessive blurring in some regions while still leave low-frequency noise in other regions. On the other hand, our method shows much higher quality denoising results across a wide range of noise levels and different characteristics of image features.

4.4 Appendix: Stochastic Error Bounding

We first introduce the t-procedure, employed for defining homogeneous pixels:

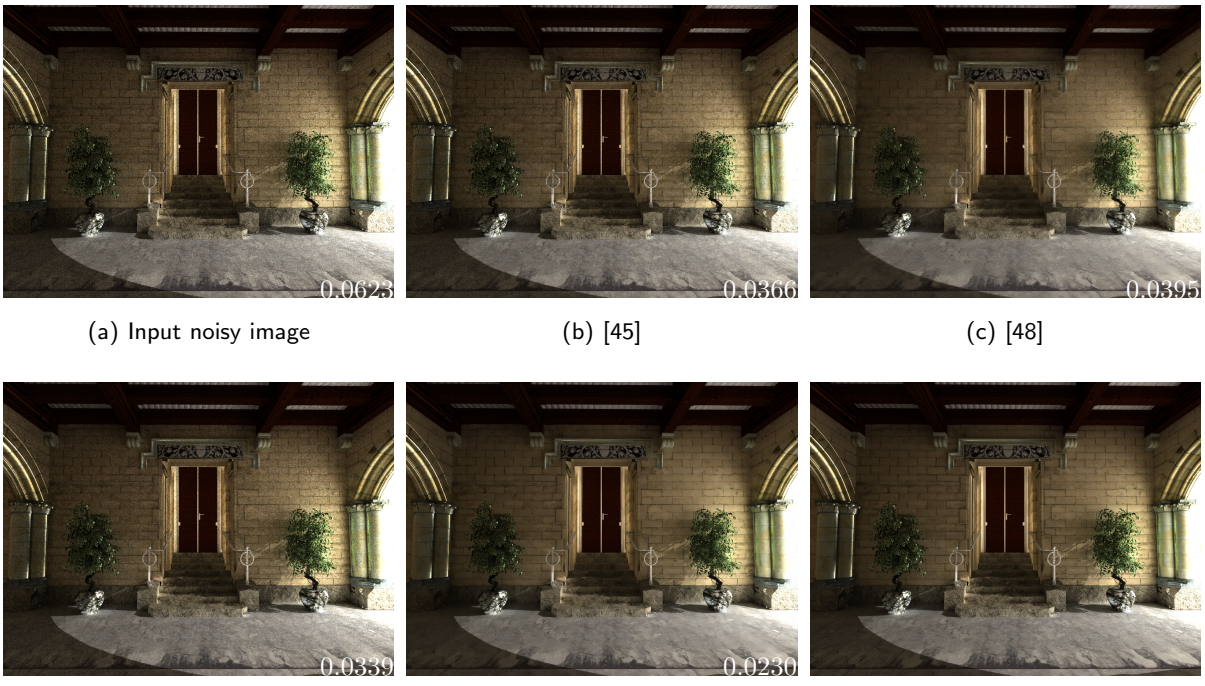
$$P\left(\bar{x}_p - L/2 \leq \mu_p \leq \bar{x}_p + L/2\right) = 1 - \alpha, \tag{4.3}$$

The equation says that the true mean μ_p at pixel p is in the confidence interval, $(\bar{x}_p - L/2, \bar{x}_p + L/2)$, with a probability of $1 - \alpha$; for the sake of simplicity, we use the term L to indicate the length of the confidence interval $\frac{2t_{\alpha/2,n_p-1}s_p}{\sqrt{n_p}}$.



(a) Input noisy i. (b) Virtual f. i. (c) Ours (d) Anisotropic

Figure 4.13: Two denoised images for the glossy metal can in the shower booth scene by our method and the anisotropic method [45].



(a) Input noisy image (b) [45] (c) [48]

(d) [83] (e) Our method (f) Reference

Figure 4.14: An input noisy image (generated with 64 ray samples) and its denoising results in the outdoor scene. The reference image is generated with 10 K ray samples per pixel. RMS errors are shown in images.

The D_p is a weight sum of values of homogeneous pixels in our denoising method, and can be represented in a simple equation as the following:

$$D_p = \sum_{p' \in H(\Omega_p)} w_{p'} \bar{x}_{p'}, \tag{4.4}$$

where $\bar{x}_{p'}$ and $w_{p'}$ correspond to the sample mean of pixel p' and its weight, respectively.

Because we select only homogeneous pixels $p' \in H(\Omega_p)$ for denoising according to Eq. (2) in the main paper, the sample means $\bar{x}_{p'}$ of the homogeneous pixels is in the following range:

$$\bar{x}_p - L/2 \leq \bar{x}_{p'} \leq \bar{x}_p + L/2. \tag{4.5}$$

By substituting Eq. (4.5) into Eq. (4.4), we have the following inequality:

$$\sum_{p' \in H(\Omega_p)} w_{p'}(\bar{x}_p - L/2) \leq D_p \leq \sum_{p' \in H(\Omega_p)} w_{p'}(\bar{x}_p + L/2),$$

where $w_{p'}$ and $\bar{x}_p + L$ are zero or positive real numbers.

Because the term $\bar{x}_p \pm L/2$ in the above equation is irrelevant to p' and thus a constant, we reach the following inequality:

$$(\bar{x}_p - L/2) \sum_{p' \in H(\Omega_p)} w_{p'} \leq D_p \leq (\bar{x}_p + L/2) \sum_{p' \in H(\Omega_p)} w_{p'}.$$

The sum of all the weights is one because of the normalization term in non-local means filtering (Eq. (1) in the main paper). As a result, we have the following inequality:

$$\bar{x}_p - L/2 \leq D_p \leq \bar{x}_p + L/2. \tag{4.6}$$

By subtracting μ_p from D_p based on Eq. (4.3) and Eq (4.6), we reach the following inequality:

$$P(-L \leq D_p - \mu_p \leq L) \geq 1 - \alpha. \tag{4.7}$$

Note that because we take a conservative value L, in this inequality, the probability should be higher than or equal to $1 - \alpha$.

The above inequality can be rewritten

$$P\left(|D_p - \mu_p| \leq \frac{2t_{\alpha/2, n_p - 1} s_p}{\sqrt{n_p}}\right) \geq 1 - \alpha. \tag{4.8}$$

□

Chapter 5. Adaptive Rendering based on Weighted Local Regression

5.1 Local Regression based Filtering

In this section we give a brief background on local regression and its application to reconstruction.

Local regression [8, 61] is a smoothing method for fitting parametric curves or surfaces, $f(\mathbf{x})$, based on a neighborhood of $\mathbf{x}$. Its underlying statistical model is:

$$y = f(\mathbf{x}) + \epsilon, \tag{5.1}$$

where y and $\mathbf{x}$ denote $\mathbb{R}^1$-valued response variable and $\mathbb{R}^D$-valued predictor variables, respectively. We also call a predictor variable a feature variable for our problem. $\mathbf{x}_j$ indicates j-th feature in its feature vector $\mathbf{x}$. In our problem features can include positions of primary rays, their corresponding textures, depths, normals, etc. The noise ϵ is modeled by additive random noise that has zero mean and bounded variance.

The unknown continuous function $f(\mathbf{x})$ can be approximated locally based on the Taylor series as follows:

$$f(\mathbf{x}) \approx f(\mathbf{x}^c) + (\mathbf{x} - \mathbf{x}^c)^T \nabla f(\mathbf{x}^c) + \frac{1}{2}(\mathbf{x} - \mathbf{x}^c)^T \mathbf{H} f(\mathbf{x}^c)(\mathbf{x} - \mathbf{x}^c) + .. \tag{5.2}$$

where $\mathbf{x}^c$ represents a center feature vector in a neighborhood of $\mathbf{x}$, and $\mathbf{H}f(\mathbf{x}^c)$ is the Hessian matrix at $f(\mathbf{x}^c)$. Eq. 5.2 can be expressed as a sum of inner products:

$$f(\mathbf{x})) \approx \alpha + \beta^T (\mathbf{x} - \mathbf{x}^c) + ... \tag{5.3}$$

A weighted least squares minimization can be formulated to determine the unknown coefficients, i.e. α, β, etc., as follows:

$$\min_{\alpha,\beta} \sum_{i=1}^{n} \left(y^i - \left(\alpha + \beta^T (\mathbf{x^i} - \mathbf{x}^c) + ...\right)\right)^2 \Pi_{j=1}^{D} w\left(\frac{\mathbf{x}_j^i - \mathbf{x}_j^c}{h\mathbf{b}_j}\right), \tag{5.4}$$

where $\mathbf{x}^i$ and y^i are i-th samples for features and their corresponding response, respectively. The term $\Pi_{j=1}^{D}(\frac{\mathbf{x}_j^i - \mathbf{x}_j^c}{h\mathbf{b}_j})$ is a commonly used multi-dimensional kernel based on the product of 1-D kernels. The kernel $w(\cdot)$ is commonly chosen to be a symmetric decreasing function (e.g. Epanechnikov kernel) for allocating high weights to close samples from $\mathbf{x}^c$. The term $h\mathbf{b}_j$ controls the bandwidth for the j-th feature vector. The bandwidth term allows control over the relative weighting between the feature dimensions. In our formulation we have chosen to separate the bandwidth component into two separate terms: $\mathbf{b}_j$ that varies per feature, and a scaling factor h that is shared across all feature types. This breakdown aids in the process of selecting optimal bandwidths for reconstruction and will be discussed in detail in Sec. 5.2. Note that the minimization of Eq. 5.4 is also used in Moving Least Squares (MLS) for surface reconstruction [49].

In our problem we use the local linear regression for reconstructing output values; as a result, unknown coefficients in Eq. 5.4 are only α and β^T. One may concern that the local regression model may be ineffective in preserving sharp edges and discontinuities. Since we consider weights computed from D

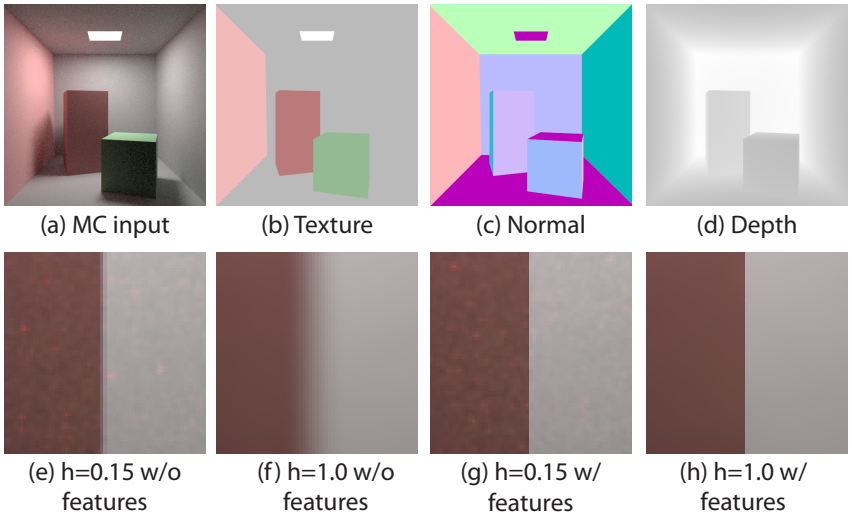


Figure 5.1: An input image (a) generated by MC ray tracing in the Cornell box. Zoomed-in images (e),(f) and (g),(h) are regression results w/o and w/ considering features (b), (c), and (d) stored in G-buffers, respectively. Image (h) demonstrates that a large filter width reconstruction with geometric features can preserve geometric discontinuities in the result, while filtering out high variance noise.

dimensional feature space, our method handles discontinuities effectively (Fig. 5.1). We have considered other alternatives such as logistic regression for handling such discontinuities as well as higher order local regression, but found that our weighted linear local regression strikes an excellent balance between denoising quality and computational efficiency while preserving feature edges. We also use a separate, quadric local regression for estimating second derivatives for different feature subspaces (Sec. 5.2.2).

Based on computed $\hat{\alpha}$ and $\hat{\beta}^T$ we can reconstruct a low order function, $\hat{f}_{h\mathbf{b}}(\mathbf{x})$, for the unknown function $f(\mathbf{x})$. For reconstructing an output value at a center sample $\mathbf{x}^c$ in our problem, we first reconstruct $\hat{f}_{h\mathbf{b}}(\mathbf{x})$ based on neighboring samples of $\mathbf{x}^i$ and y^i with the filtering bandwidth $h\mathbf{b}$, and then use $\hat{f}_{h\mathbf{b}}(\mathbf{x}^c)$, which is $\hat{\alpha}$, as the final reconstruction output value for $\mathbf{x}^c$.

Normal equation. Given a selected kernel function $w(\cdot)$ and bandwidths $h\mathbf{b}$ for feature types, the weighted least squares optimization problem shown in Eq. 5.4 has the following closed form solution, i.e. normal equation:

$$[\hat{\alpha}, \hat{\beta}]^T = (X^T W X)^{-1} X^T W Y, \tag{5.5}$$

where X is the $n \times (D+1)$ design matrix whose i-th row is set as $[1, (\mathbf{x}^i - \mathbf{x}^c)^T]$, and Y is the $n \times 1$ output vector $Y = [y^1, ..., y^n]^T$. n is the number of samples $\mathbf{x}^i$ within a squared window defined from the center sample $\mathbf{x}^c$. The matrix W is $n \times n$ diagonal matrix, whose elements are defined by the weight function $w(\frac{\mathbf{x}^i_j - \mathbf{x}^c_j}{h\mathbf{b}_j})$.

Intuitively speaking, the bandwidths $h\mathbf{b}_j$ act as smoothing parameters; controlling the trade-off between bias and variance of our local regression based reconstruction method. Large values of $h\mathbf{b}_j$ produce smooth reconstructions with low variance, but suffer from a high bias because of combining samples over a large spatial extent. Conversely small values of $h\mathbf{b}_j$ produce low bias, but resulting regression can be quite noisy. We present our optimization process in Sec. 5.2 to choose bandwidth parameters such that we minimize the Mean Squared Error (MSE), $E[(\hat{f}_{h\mathbf{b}}(\mathbf{x}) - f(\mathbf{x}))^2]$, of estimated low order functions $\hat{f}_{h\mathbf{b}}(\mathbf{x})$.

The normal equation unfortunately can be unstable and requires a high computation cost, as we

Algorithm 1 Local Regression based Adaptive Rendering

Input: Feature vector $\mathbf{x}^i$ and intensity y^i generated by MC rendering

Output: Sampling map for the next rendering pass or final output

```
for x^c in each pixel do
    Estimate b_j based on second partial derivatives (Sec. 5.2.2)
    for each h ∈ {h_min, h_max} do
        Compute f̂_{hb}(x^c) using our truncated SVD (Sec. 5.2.4)
        Estimate bias and variance of f̂_{hb}(x^c) (Sec. 5.2.1)
    end for
    Compute h_opt using parametric error analysis (Sec. 5.2.3)
    Compute f̂_{h_opt b}(x^c) (Sec. 5.2.4)
end for
if Sampling budget remains then
    return a sampling map based on ΔrMSE(x) (Sec. 5.3)
else
    return f̂_{h_opt b}(x^c)
end if
```

consider more types of features. Instead of using the normal equation we propose a truncated SVD to robustly solve our weighted least squares optimization problem (Sec. 5.2.4).

By using local regression and performing error analysis for our problem in a principled way, our approach has a number of significantly different characteristics over prior methods. First, our method locally reconstructs a parametric function for samples and minimizes its reconstruction error more robustly over prior approaches [59, 60, 41] that perform point estimation on $f(\mathbf{x})$. Second, our method can measure importance of different types of features by estimating second partial derivatives for features and even ignore noisy input features. Finally, our error analysis is naturally combined with sampling, to guide available ray sample budgets.

5.2 Adaptive Reconstruction

In this section we explain our adaptive reconstruction approach. We start by formalizing the goal of our reconstruction method and provide the high level overview of our algorithm (Sec. 5.2.1). We formulate a novel optimization process for our regression scheme that selects optimal filter bandwidths, which reduce the estimated error (Sec. 5.2.2 and Sec. 5.2.3). Sec. 5.2.4 describes our novel regression strategy, which is capable of handling noise coming from both input predictor variables as well as response variables. This results in a high quality reconstruction that is robust against noise and generates smooth images, while preserving salient edges.

In our problem we assume that all the required information for our reconstruction are provided by MC ray tracing, and the information is stored in each pixel. Specifically, given a center pixel $\mathbf{x}^c$ we define its neighboring pixels as $\mathbf{x}^i$ in a filtering window. A pseudocode of our overall algorithm is provided in Algorithm 1.

5.2.1 Optimization Goal

The optimization goal of our local regression based filtering is to minimize MSE, which consists of the bias and variance terms: $MSE = bias(\hat{f}_{h\mathbf{b}}(\mathbf{x}))^2 + var(\hat{f}_{h\mathbf{b}}(\mathbf{x}))$ by adjusting reconstruction bandwidths $h\mathbf{b}_j$ in Eq. 5.4, where h is a shared bandwidth for modulating the magnitude of all the feature bandwidths and $\mathbf{b}_j$ controls j-th feature bandwidth.

One can easily show that our weighted local regression is a linear smoother in the output variable. The estimation result $\hat{f}_{h\mathbf{b}}(\mathbf{x})$ for a pixel $\mathbf{x}$ under our regression model can be then expressed by $\hat{f}_{h\mathbf{b}}(\mathbf{x}) = \sum_{i=1}^{n} l^i_{h\mathbf{b}}(\mathbf{x}) y^i$, where $l^i_{h\mathbf{b}}(\mathbf{x})$ is a weight for the noisy output y^i given bandwidth values $h\mathbf{b}_j$.

The bias of $\hat{f}_{h\mathbf{b}}(\mathbf{x})$ is then reformulated as follows:

$$\begin{aligned} E\left(\hat{f}_{h\mathbf{b}}(\mathbf{x}) - f(\mathbf{x})\right) &= \sum_{i=1}^{n} l^i_{h\mathbf{b}}(x) E(y^i) - f(\mathbf{x}) \\ &= \sum_{i=1}^{n} l^i_{h\mathbf{b}}(\mathbf{x}) f(\mathbf{x}^i) - f(\mathbf{x}) \ (\because \text{Eq. 5.1}) \\ &= \sum_{i=1}^{n} l^i_{h\mathbf{b}}(\mathbf{x}) y^i - y. \end{aligned} \tag{5.6}$$

The observed values y^i and y from the pixel filter (e.g. box filter) provided by users are typically considered as unbiased estimation of the unknown $f(\mathbf{x}^i)$ and $f(\mathbf{x})$ [59]. As a result, the unbiased estimation for the bias term is then achieved as $\sum_{i=1}^{n} l^i_{h\mathbf{b}}(\mathbf{x}) y^i - y$, when we plug-in y^i and y to the unknown $f(\mathbf{x}^i)$ and $f(\mathbf{x})$, respectively, i.e. the last step of Eq. 5.6. The variance of our reconstruction method can be represented in a similar manner:

$$var(\hat{f}_{h\mathbf{b}}(\mathbf{x})) = \sum_{i=1}^{n} {l^i_{h\mathbf{b}}}^2(\mathbf{x}) var(y^i), \tag{5.7}$$

where $var(y^i)$ is the variance of the sample mean at $\mathbf{x}^i$.

The bias and variance terms of our optimization goal have the well-known asymptotic expressions [61]. The bias term has the following asymptotic relationship:

$$bias(\hat{f}(\mathbf{x})_{h\mathbf{b}}) \propto \frac{1}{2} h^2 trace(B\mathbf{H}\hat{f}_{h\mathbf{b}}(\mathbf{x})), \tag{5.8}$$

where a diagonal matrix B is set as $diag(\mathbf{b}_1^2, ..., \mathbf{b}_D^2)$, $trace(.)$ is the matrix trace, and $\mathbf{H}\hat{f}_{h\mathbf{b}}(\mathbf{x})$ is the Hessian matrix. In addition,

$$var\left(\hat{f}_{h\mathbf{b}}(\mathbf{x})\right) \propto \frac{1}{n(\mathbf{x}) h^D \Pi_{j=1}^{D} \mathbf{b}_j}, \tag{5.9}$$

where $n(\mathbf{x})$ is the number of samples in a local neighborhood of $\mathbf{x}$. For example, $n(\mathbf{x})$ simply counts the number of samples in pixel $\mathbf{x}$ in our reconstruction.

Given our optimization goal represented with bias and variance terms (Eq. 5.6 and Eq. 5.7), we seek a set of bandwidths $h\mathbf{b}_j$ that reduce MSE of each pixel; h and $\mathbf{b}_j$ are defined for each pixel $\mathbf{x}$, but we do not explicitly encode that information in their notations for simplicity. A straight-forward approach for choosing optimal bandwidths $h\mathbf{b}_j$ for all the features given our optimization goal is to test all the possible combinations of bandwidth values within a user-defined set (e.g. $h\mathbf{b}_j \in \{0.1, 0.3, 0.5, 0.7, 0.9\}$). This brute-force approach unfortunately requires an exponentially growing number of tests, as we consider more features for high quality reconstruction.

Instead we propose a two-step optimization process that constructs parametric curves for bias and variance terms, and has a linear time complexity as the feature dimension increases. At a high level we

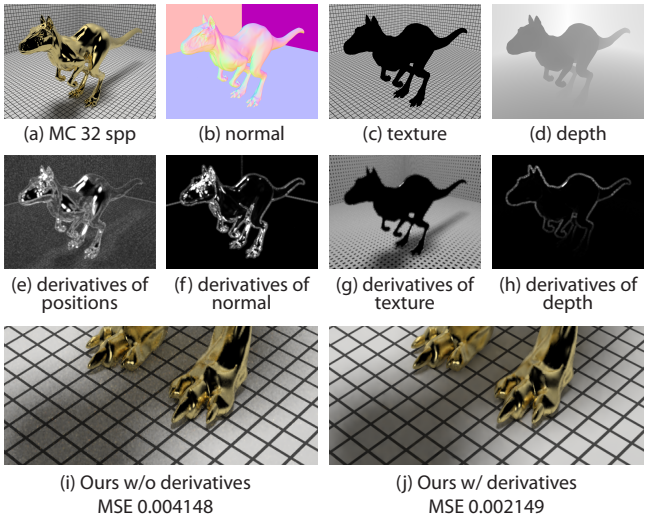


Figure 5.2: Input image (a) is generated with 32 uniform spp, and its normal (b), texture (c), and depth(d) images are visualized. The estimated second partial derivatives are shown from (e) to (h). The results (j) and (i) are generated w/ and w/o considering derivative information, respectively. Using different filtering bandwidths based on the partial derivatives produces both more visually pleasing and numerically accurate results.

first estimate our per-feature bandwidths $\mathbf{b}_j$ by estimating their corresponding second partial derivatives (Sec. 5.2.2). We then optimize the shared bandwidth h given the estimated feature bandwidths (Sec. 5.2.3).

One may attempt to directly use the asymptotic expressions for estimating optimal bandwidths. This approach is referred to as plug-in bandwidth selection [42]. The caveat of this approach is the poor behavior with a small number of samples, because the asymptotic expression itself assumes a large number of samples. Our key idea is to utilize only functional relationships between $\mathbf{b}_j$ and second partial derivatives (Sec. 5.2.2), also between h and errors (Sec. 5.2.3).

5.2.2 Estimating Feature Bandwidths $\mathbf{b}_j$

The asymptotic bias term (Eq. 5.8) consists of multiple bias terms, each of which is introduced from each feature subspace. The amount of bias caused by considering j-th feature $\mathbf{x}_j$ is proportional to $(h\mathbf{b}_j)^2|\frac{\partial^2 f}{\partial \mathbf{x}_j^2}|$. As shown in Fig. 5.2, second partial derivatives of features can vary a lot. Intuitively, selecting a small bandwidth for a feature with a strong second partial derivative is desirable to reduce its corresponding bias, according to the asymptotic bias expression. Small bandwidths, however, lead to an increase in the asymptotic variance expression (Eq.5.9). In other words, the second derivatives tell us the relative impact that each feature has on our reconstruction error. In this section we estimate initial values for $\mathbf{b}_j$ based on relative magnitudes of their corresponding second derivatives, followed by adjusting magnitudes of filtering widths by optimizing h in the next section.

Since $\mathbf{b}_j$ has a linear relationship with $|\frac{\partial^2 f}{\partial \mathbf{x}_j^2}|^{-0.5}$ according to the asymptotic bias equation, we choose $\mathbf{b}_j$ to be $|\frac{\partial^2 f}{\partial \mathbf{x}_j^2}|^{-0.5}$. Given this equation, our next goal is to estimate the second partial derivatives of the unknown function $f(\mathbf{x})$.

One of the fundamental benefits of local regression is that we can easily estimate second partial derivatives using quadratic local regression at each pixel, since analytic second derivatives are directly

computable from the quadratic solution form. In order to perform local quadratic regression we also need to select another set of bandwidths for the local quadratic regression; note that the bandwidths required for performing the quadratic fit are different from $h\mathbf{b}_j$ used for the linear local regression that we aim to optimize. Estimating these additional bandwidths for the local quadratic regression is far more difficult than what we are trying to address, since it is related to estimating higher order derivatives [61]. In addition, estimating second derivative information given samples can be quite noisy, compared to estimating the unknown function $f(\mathbf{x})$. To solve both issues, we choose to use a large single bandwidth, e.g., one, on all the features for the quadratic fit. Since the estimated quadratic fit is computed with such a large bandwidth, its second derivatives become smooth.

Fig. 5.2 shows denoising results w/ and w/o estimating partial derivatives and thus using separate feature bandwidths in the killeroo benchmark. Considering feature bandwidths gives not only visually pleasing, but also numerically accurate results compared to using the same bandwidth for all the feature subspaces.

5.2.3 Parametric Error Estimation

Given feature bandwidths $\mathbf{b}_j$ estimated in the last section we now focus on optimizing the shared bandwidth h, in terms of minimizing MSE. Our error analysis first fits parametric curves for bias and variance terms of MSE, as a function of the shared bandwidth parameter h, and then analytically optimizes h to minimize MSE.

As the first step for optimizing h we apply our linear local regression method with different bandwidth values for h in a range of $[h_{min}, h_{max}]$. This process provides us pairs of h value and its corresponding bias value according to the unbiased bias estimator shown in Eq. 5.6; note that we do not use the asymptotic bias equation for this process.

As the second step we fit a curve based on ordinary least squares with observed pairs, to generate a parametric bias function, $bias_h(\mathbf{x})$, as a function of h. To use the ordinary least squares, a proper model for $bias_h(\mathbf{x})$ should be chosen in terms of h. For the parametric bias function, we employ the functional relationship between bias and h shown in the asymptotic bias equation (Eq. 5.8). As a result, we use the following quadratic form for the underlying model of $bias_h(\mathbf{x})$:

$$bias_h(\mathbf{x}) = \lambda_1 h^2. \tag{5.10}$$

The coefficients λ_1 then can be easily estimated by the ordinary least squares.

To construct a parametric curve for the variance term, we also utilize the functional relationship captured in the asymptotic variance expression (Eq. 5.9). One modification of the expression to our method is to change the input dimension D into the local dimension d, which will be computed by our truncated SVD explained in the next section. A polynomial model of the variance term is set as:

$$var_h(\mathbf{x}) = (\kappa_0 + \frac{\kappa_1}{h^d})\frac{1}{n(\mathbf{x})}. \tag{5.11}$$

The coefficients κ_0 and κ_1 are computed also using the ordinary least squares.

Based on estimated two parametric curves $bias_h(\mathbf{x})$ and $var_h(\mathbf{x})$, and feature bandwidths $\mathbf{b}_j$, we can parameterize our MSE as $MSE_h(\mathbf{x})$, with respect to the shared bandwidth h. Since two parametric curves are monotonic functions as a function h, one can analytically compute the optimal bandwidth as $h_{opt} = \frac{d\kappa_1}{4\lambda_1^2 n(\mathbf{x})}^{\frac{1}{d+4}}$ based on $\frac{\partial MSE_h(\mathbf{x})}{\partial h} = 0$. In addition, an estimated MSE of the reconstructed image with h_{opt} is computed by $MSE_{h_{opt}}(\mathbf{x})$.

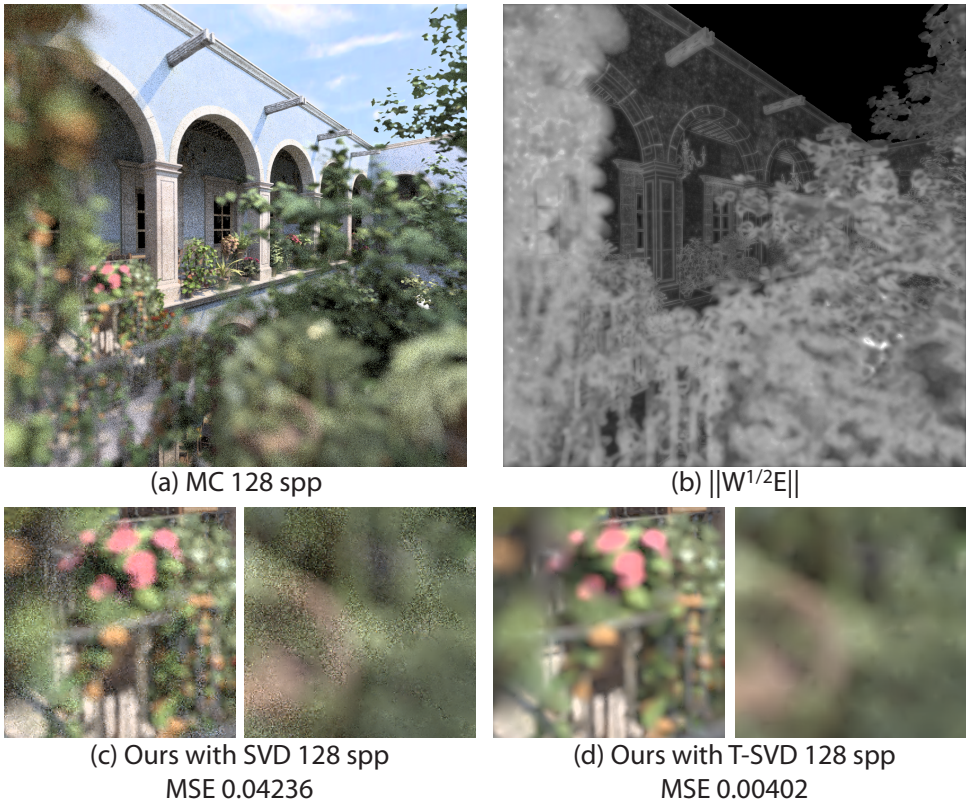


Figure 5.3: This figure shows reconstruction results w/ and w/o performing our truncated SVD (T-SVD). The input image (a) is a zoomed-in defocused region of the San Miguel scene, whose features are very noisy. We also show the spectral norm $\|W^{1/2}E\|$.

Fig. 5.5 shows an estimated MSE of our reconstructed image generated from four spp in the killeroo-gold scene. It matches well with its ground truth MSE, even though the estimated MSE is generated with four spp.

5.2.4 Truncated SVD based Local Regression

Common local regression methods assume that input predictor variables are not noisy, while response variables have noise. Unfortunately, our feature vectors can also carry a significant amount of noise. To address this problem we construct a reduced feature space using a truncated SVD for solving Eq. 5.4.

To solve the optimization problem (Eq. 5.4) at every pixel, we use singular value decomposition (SVD) rather than the normal equation shown in Eq. 5.5. Although SVD is considered more expensive than directly solving the normal equation it provides us with an elegant means to robustly handle rank deficient systems commonly encountered. We leverage the property of SVD rank determination to reduce our problem to a lower dimensional subspace. When combined with perturbation theory, we are able to pre-filter the matrix space for reducing noise that would lead to poor conditioning and failed reconstruction. Lastly we use the reduced dimension as a part of our adaptive sampling strategy to better allocate our ray samples (Sec. 5.3).

SVD on a $n \times p$ matrix Z leads to a factorization $Z = USV^T$, where U and V are $n \times n$ and $p \times p$ unitary matrix, respectively. Also, S is the $n \times p$ diagonal matrix, where diagonal entries, σ_m, are ordered singular values in the non-increasing order.

Typically very small singular values close to zero are suppressed for avoiding precision errors introduced on Z. This process is known as the compact SVD that gives numerically stable solutions. This approach, unfortunately, does not work well for our problem, because some column spaces of Z can be very noisy or Z can be a rank-deficient matrix. For example, distribution effects (e.g. defocusing and motion blurring) can make some column spaces in the design matrix X noisy due to noisy geometry buffers. Also, in many cases, textures and normals of samples can be same locally. This makes Z to be rank-deficient. This motivates us to use the truncated SVD based on perturbation theory [27].

The approach behind truncated SVD is to filter out small singular values less than a threshold τ by setting them to be zero. The observation behind using truncated SVD is that small singular values are often a result of corruption from noise, and these small corruptions lead to significant changes in the least squares solutions [27]. To adaptively select the threshold τ, we rely on perturbation theory that provides us with a principled approach for selecting an appropriate threshold value of τ.

In the view of the perturbation theory, a perturbation matrix Z is expressed by $Z = W^{1/2}X = W^{1/2}(\mu(X) + E)$, where $\mu(X)$ is the ground truth design matrix and E is the perturbation matrix containing error and noise. We then have the following Weyl's perturbation bound [70]:

$$|\sigma_m - \hat{\sigma}_m| \leq \|W^{1/2}E\|_2, \tag{5.12}$$

where σ_m and $\hat{\sigma}_m$ are the m-th singular value of $W^{1/2}\mu(X)$ and Z, respectively. Also, $\|W^{1/2}E\|_2$ is the spectral norm of $W^{1/2}E$.

In our problem $\mu(X)$ can be obtained with an infinite number of ray samples, and E indicates the perturbation structure on X introduced by MC ray tracing. By the central limit theorem and independence property of MC ray samples [1], we can estimate the noisy matrix E. Specifically, we assume each element in X follows a normal distribution, $X_{ij} \sim \mathcal{N}\left(\mathbf{x}_j^i - \mathbf{x}_j^c, var(\mathbf{x}_j^i) + var(\mathbf{x}_j^c)\right)$. We then model each element in E as $E_{ij} = \left(var(\mathbf{x}_j^i) + var(\mathbf{x}_j^c)\right)^{1/2}$. We estimate the unknown variances $var(\mathbf{x}_j^i)$ and $var(\mathbf{x}_j^c)$ as the sample variances of j-th feature mean at pixel $\mathbf{x}^i$ and center pixel $\mathbf{x}^c$, respectively.

The Weyl's perturbation bound in Eq. 5.12 gives us upper and lower bounds for a range of σ_m given observed values $\hat{\sigma}_m$ and $W^{1/2}E$. According to these lower and upper bounds, potential values for σ_m can be negative or zero, when $\hat{\sigma}_m$ is equal to or lower than $\|W^{1/2}E\|_2$. Since σ_m cannot be negative or zero, we set τ conservatively to be $\|W^{1/2}E\|_2$.

After performing the SVD and identifying k biggest singular values by filtering out small ones based on perturbation theory, we then approximate the matrix Z as those k largest singular values and corresponding singular vectors. This can be expressed by a compact form $Z = W^{1/2}X \approx U_k S_k V_k^t$, where U_k and V_k are $n \times k$ and $k \times k$ reduced unitary matrix respectively, and S_k is the diagonal singular matrix that has k non-zero singular values. This k-rank approximation of Z has been known as the best approximation among k-ranked matrices, according to the Schmidt approximation theorem [70]. In our problem $\|W^{1/2}E\|$ tends to monotonically decrease as a function of the number of samples. As a result, the bias introduced by the k-rank approximation goes to zero.

In summary the solution of the optimization problem of Eq. 5.4 is then represented by the truncated SVD approximation as follows:

$$[\hat{\alpha}, \hat{\beta}]^T = V_k S_k^+ U_k^T W^{1/2} Y, \tag{5.13}$$

where S_k^+ is the Moore-Penrose pseudoinverse of S_k, and is set as $[\sigma_1^{-1}, ..., \sigma_k^{-1}]$.

[1] It is well known that even with a finite number of samples, the normal distribution assumption on MC samples is a good approximation [25].

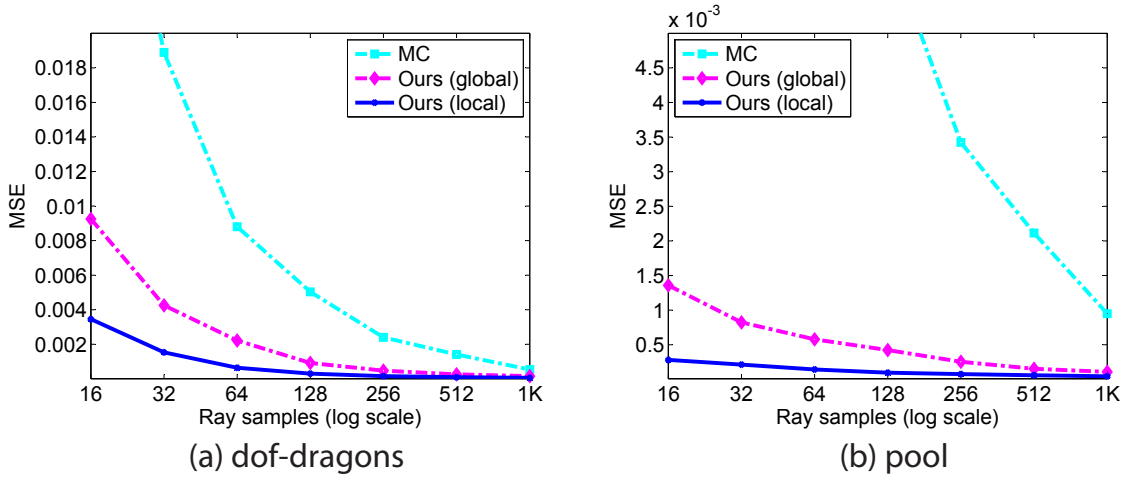


Figure 5.4: Convergence results on dof-dragons and pool scenes when we set d to the rank k of our truncated SVD (local) or to be D (global).

Fig. 5.3 shows reconstruction results computed by using the compact SVD or our truncated SVD. By using truncated SVD, we can effectively ignore noisy feature subspaces and improve the overall quality. As will be disucssed in Sec. 5.4 we use three feature types consisting of 9 dimensional feature space for all the results. Based on the proposed truncated SVD, we reduce it to a much smaller local space, which is in an approximate range of $[2.1, 3.1]$ for differnt scenes, leading to a much faster error reduction.

5.3 Adaptive Sampling

We use a common iterative approach [59, 41] to allocate available ray samples to regions with high errors. As an initial iteration we uniformly distribute a small number of ray samples, i.e. four ray samples per pixel (spp). In subsequent iterations we predict an error reduction $\Delta MSE(\mathbf{x})$ for a pixel $\mathbf{x}$, when the pixel would receive one additional sample. The main difference in our method over the common iteration sampling process is that we use our error metric, which is dependent on the local, reduced feature subspace d, not the original feature space D.

Since we perform our reconstruction with the computed optimal shared bandwidth h_{opt} its reconstruction error is estimated as $MSE_{h_{opt}}(\mathbf{x})$, where $h_{opt} = \frac{d\kappa_1}{4\lambda_1^2 n(\mathbf{x})}^{\frac{1}{d+4}}$.

$\Delta MSE(\mathbf{x})$ is defined as the reduction of MSE when the pixel $\mathbf{x}$ receives a new sample and is reconstructed again with a new optimal filtering bandwidth, say $h_{opt'}$, with the new ray sample count, $n(\mathbf{x})+1$. Since it is infeasible to construct the new optimal filtering bandwidths, we simply set $h_{opt'}$ to $\frac{d\kappa_1}{4\lambda_1^2(n(\mathbf{x})+1)}^{\frac{1}{d+4}}$. $\Delta MSE(\mathbf{x})$ is then approximated as: $MSE_{h_{opt}}(\mathbf{x}) - MSE_{h_{opt'}}(\mathbf{x})$.

In order to consider human visual perception that is more sensitive to darker areas we use the relative MSE [41], dubbed rMSE, and then $\Delta rMSE(\mathbf{x})$ is defined as the following: $\Delta rMSE(\mathbf{x}) = \frac{MSE_{h_{opt}}(\mathbf{x}) - MSE_{h_{opt'}}(\mathbf{x})}{\hat{f}_{h_{opt}} + \gamma}$, where γ is used to avoid the divide-by-zero and is set to be 0.001 in practice. We then set a number of samples, $\Delta n(\mathbf{x})$, for a pixel $\mathbf{x}$, according to its relative reduction rate over the sum from all the pixels. In other words, $\Delta n(\mathbf{x}) = \frac{\Delta rMSE(\mathbf{x})}{\sum_t \Delta rMSE(\mathbf{x}^t)}$. We generate $\Delta n(\mathbf{x})$ samples for the pixel $\mathbf{x}$ by low discrepancy sampling, a common practical choice.

A noticeable difference, however, is that we allocate ray budgets by considering the reduced d-dimensional feature space, not the D-dimensional original feature space, because $\Delta rMSE(\mathbf{x})$ is a function

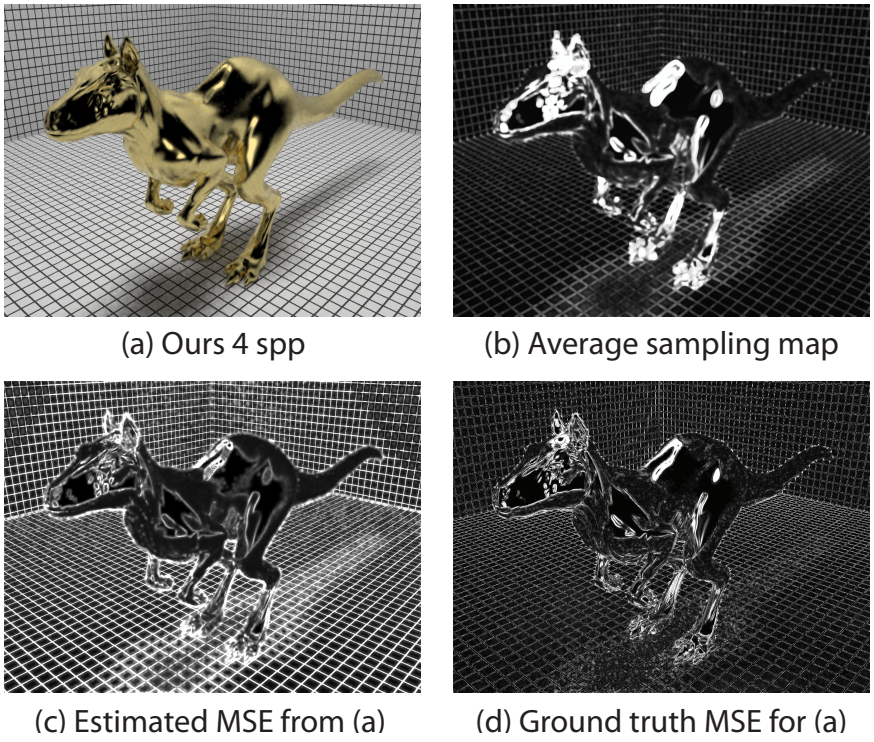


Figure 5.5: The estimated MSE (c) generated from four spp (a) matches reasonably well with its ground truth (d) computed between (a) and its reference. The average sampling map (b) has a similar trend with the ground truth, demonstrating that our method generates samples to effectively reduce MSE.

of d, computed based on the truncated SVD. One can easily show that bias and variance terms of our method with h_{opt} have the same reduction rate of $n(\mathbf{x})^{-\frac{4}{d+4}}$; deriving these terms can be done by putting h_{opt} into bias and variance terms $bias_h(\mathbf{x})$ and $var_h(\mathbf{x})$ fitted in Sec. 5.2.3. $MSE(\mathbf{x})$ of our reconstruction method in turn reduces in the same rate. Theoretically this can lead to a better convergence ratio than considering all the feature types. For example, in a benchmark of dof-dragons (Fig. 5.7), the average d across all the image pixels generated with 32 spp is 2.4, which is quite lower than the original dimensionality, 9. Fig. 5.4 empirically verifies our theoretic results by showing convergence graphs when we set d to be the rank k of the truncated SVD or D the dimensionality of the original feature space.

Fig. 5.5 shows an average sampling map of all the sampling maps that our method have generated to reach 32 spp from 4 spp, as well as the ground truth of $\Delta rMSE(\mathbf{x})$ by measuring the difference between the reference image and reconstructed one with four spp. The average sampling map shows a similar trend, but in a conservative way, with the ground truth MSE, demonstrating that our method effectively reduces MSE as we generate more samples.

5.4 Results and Comparisons

We have implemented our method on top of a well-known rendering system, pbrt2 [54]. For all the tests we use an Intel i7-3960X CPU machine with 3.3 GHz, and NVIDIA GeForce GTX 580 for a CUDA implementation of our reconstruction method. Thanks to the nature of our image-space adaptive reconstruction method, it is trivial to implement our method on a GPU.

We select box filter with a small width (e.g. 0.5) for the pixel filter, which is a de facto standard [54].

Figure 5.6: Equal-time adaptive rendering results in the San Miguel scene, a challenging benchmark containing highly complex geometry and textures with a strong depth-of-field effect. The image (b) is generated by 128 uniform low discrepancy samples per pixel (spp). NLM [Rousselle et al. 2012] leaves high-frequency noise on both focused (the top row) and defocused (the bottom row) regions. SURE [Li el al. 2012] provides relatively better results on the focused region, but creates noticeable artifacts on the defocused regions. Our method shows more visually pleasing results on both regions and numerically better, more than 3:1 reduction ratios on average, over NLM and SURE.

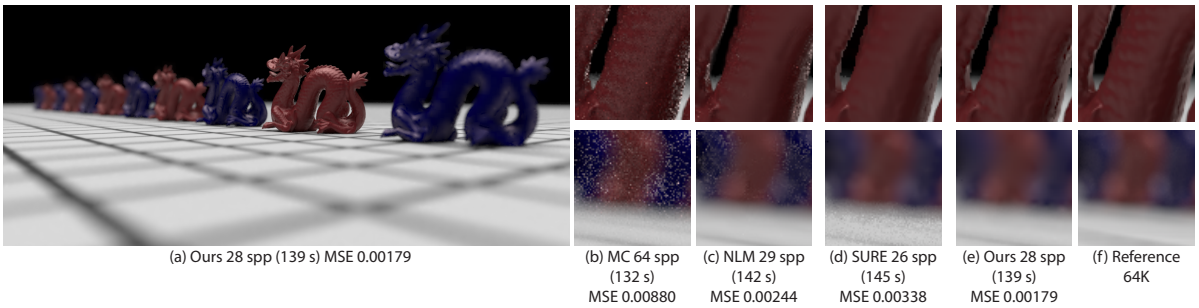


Figure 5.7: Equal-time comparisons in the dof-dragons. Previous methods show either over-smoothed results or noise. Our method not only preserves the detailed geometry, but also provides smoothed results on the defocused area.

We use the well-known bounded kernel, bisquare kernel $w(t) = \frac{15}{16}(1-t^2)^2$ for $|t| < 1$, as the kernel function in Eq. 5.4, since it gives more smooth and visually pleasing results than the Epanechnikov kernel, and we also use a 19 x 19 filtering window size for our reconstruction, since we found that it gives a good balance between performance and quality. We consider four feature types resulting in a 9 dimensional space: 2D coordinates, 3D normals, 1D depths, and 3D textures of primary rays. Features are commonly normalized [63], and we also normalize them in a range of $[0, 1]$. We test five different values for h in the range of $[h_{min}, h_{max}]$, as mentioned in Sec. 5.2.3. These five values are defined as the following: $[0.2 * h_{max} = h_{min}, 0.4 * h_{max}, 0.6 * h_{max}, 0.8 * h_{max}, 1.0 * h_{max}]$. In the initial pass, h_{max} is set as a large value, 1. For subsequent iterations, we adaptively select h_{max} based on $\omega * h_{opt}(\mathbf{x})$, where $h_{opt}(\mathbf{x})$ is the optimal value computed in the previous iteration. ω is a user-defined constant and set to 1.5, since it gives a good trade-off between performance and quality of our reconstruction method.

We compare our method with the state of the art adaptive rendering methods [60, 41]. We use five iterations of adaptive sampling to reach the target sample count for all the tests, and four ray samples are uniformly distributed in the first iteration. For the non-local means (NLM) adaptive rendering [60], we use the CUDA code provided by the authors. We use their suggested user parameters (i.e. 7 x 7 patch size, and damping factor $k = 0.45$) for all the scenes. According to their paper, NLM is tested with a 41 x 41 filtering window size for the conference scene, to more aggressively smooth out outliers, while using a 21 x 21 filtering window size for the other scenes; note that our method uses a single window size (19 x 19) across all the scenes to demonstrate the robustness of our method. We have implemented cross bilateral filtering based on Stein's unbiased risk estimator (SURE) as described in Li et al. [41]. This method is not limited to a specific filtering method, but the authors recommended the cross bilateral filtering, because it shows the best balance between performance and quality. We have communicated with the authors and used the suggested standard deviation parameters (e.g. 0.8 for normal, 0.25 for texture, and 0.6 for the normalized depth). As suggested by the authors, we prefilter their estimated MSE image using the cross bilateral filtering, and the prefiltered values are used for both their bandwidth selection and sampling map. These parameters are used for all the scenes.

As a quantitative measure for comparisons we use the relative MSE [59] that is computed as an average of $(\hat{f}_h(\mathbf{x}) - f(\mathbf{x}))^2/(f(\mathbf{x})^2 + 0.01)$, where 0.01 is a parameter for avoiding dividing by zero. The relative MSE gives more penalty for errors caused in dark areas.

Benchmarks. To test behaviors of all the tested methods in a variety of rendering effects we have chosen the following well-known benchmarks: 1) San Miguel (1024 x 1024), 2) killeroo-gold (1368 x 1026), 3) dof-dragons (1500 x 636), 4) pool (1024 x 1024), and 5) conference scene (1024 x 1024). The number in each parenthesis denotes the image resolution used for each scene, and is adopted from pbrt scene setting. The San Miguel scene has complex textured geometry, and a large portion of its image is defocused due to strong depth-of-field (DOF) effects. The DOF on complex geometry is a challenging benchmark for filtering methods using geometric information, because their corresponding G-buffers are quite noisy. In the killeroo-gold, the killeroo model has the highly glossy gold material that makes spike noise and strong highlights. The dof-dragons scene consists of instanced multiple dragons with complex geometry, and each dragon has very different DOF effects. In the pool scene each billiard ball with a plastic material has a different motion that leads to different motion blurs. The conference scene has glossy materials that introduce many spike noise. All the scenes are rendered by path tracing.

Comparisons. Overall our method shows visually pleasing and numerically better results over NLM and SURE, while these two prior methods show over-blurring details in some regions with leaving out high-frequency artifacts in other regions. Specifically, in the San Miguel scene (Fig. 5.6), we conduct

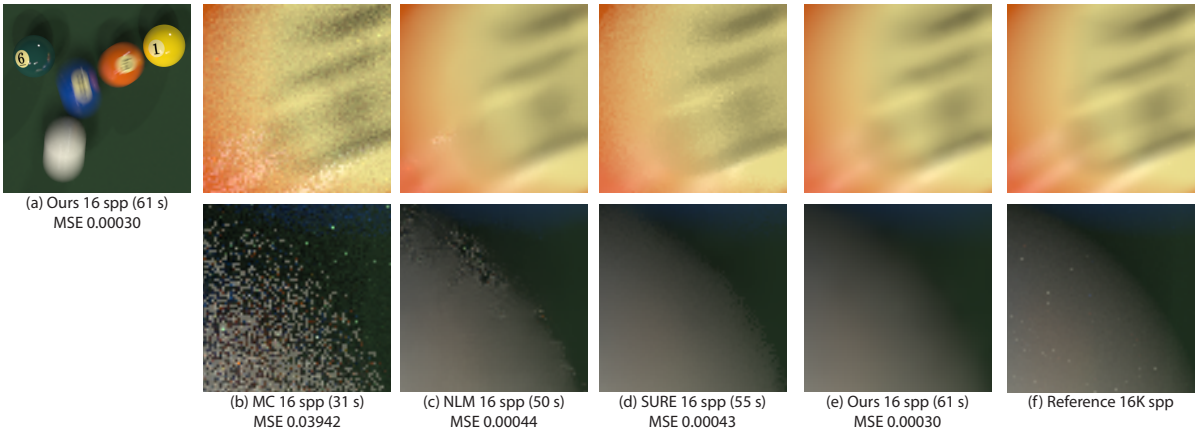


Figure 5.8: Quality comparisons with the equal sample count in the pool scene. Our method (a) and (e) shows visually better and a lower MSE compared to previous methods (c), and (d). Although our method is 22% slower than NLM, our method achieves 46% lower MSE than NLM. Even 16K spp without any reconstruction (f) exhibit spike noise. Our method with 16 spp shows a better MSE than uniform sampling with 1 k spp according to Fig. 5.4-(b).

equal-time comparisons. Overall our method shows more visually pleasing and numerically accurate results, more than 3:1 reduction ratios on average, compared to NLM and SURE. The NLM leaves out many high frequency artifacts on both focused (the first row) and defocused regions (the second row). Because NLM does not utilize G-buffers (i.e. normal, depth, and textures), it shows noisy results, when the input color information is highly noisy. While NLM achieves a high reconstruction speed without considering any features, ignoring them is the main weakest of NLM. On the other hand, SURE provides a relatively better filtering result in the focused region, where the G-buffers can guide some edge information, but noticeable artifacts are generated in the defocused region, where the geometry is not helpful. SURE attempted to ameliorate this problem by applying the prefiltering using a cross bilateral filtering, to compute smooth filtering bandwidths and sampling map. However, the prefiltering fails to smooth out some pixels where the geometry is highly noisy, and spike noise remains, since the energy of spike noise is not well distributed. In addition, a few null radiance pixels remained, when they failed to generate more rays on the pixels where initial MC ray samples do not find any light paths.

In the killeroo-gold scene (Fig. 5.11), we conduct the equal-time comparisons. Overall our method shows a visually pleasing result and the lowest MSE. Specifically, our method shows 10:1 and 2:1 MSE reduction ratios over SURE and NLM, respectively. For highlights of the leg NLM produces under-blurred results and SURE generates over-blurred results.

We have measured convergence rates of the relative MSE in the San Miguel and killeroo-gold scenes. As shown in Fig. 5.9, our method consistently shows lower MSEs across all the ray sample counts over NLM and SURE. In the killeroo, the MSE reduction of SURE is very poor although their method produces a quite smooth filtered image. Unfortunately, the uniform sampling with higher than 70 spp shows even lower MSE than SURE in the killeroo-gold. Specifically, the strong highlights on the killeroo are not preserved well, which gives a high error. Technically, this error is mainly from bias, and smaller bandwidths are required for reducing the bias. Unfortunately, the bandwidth for each feature in SURE is set as a globally fixed user parameter. One may reduce the user parameter to reduce such bias, but it typically generates more noise on other areas. On the other hand, NLM is outperformed by SURE in the San Miguel with around 100 spp. This is mainly because NLM does not consider geometric features,

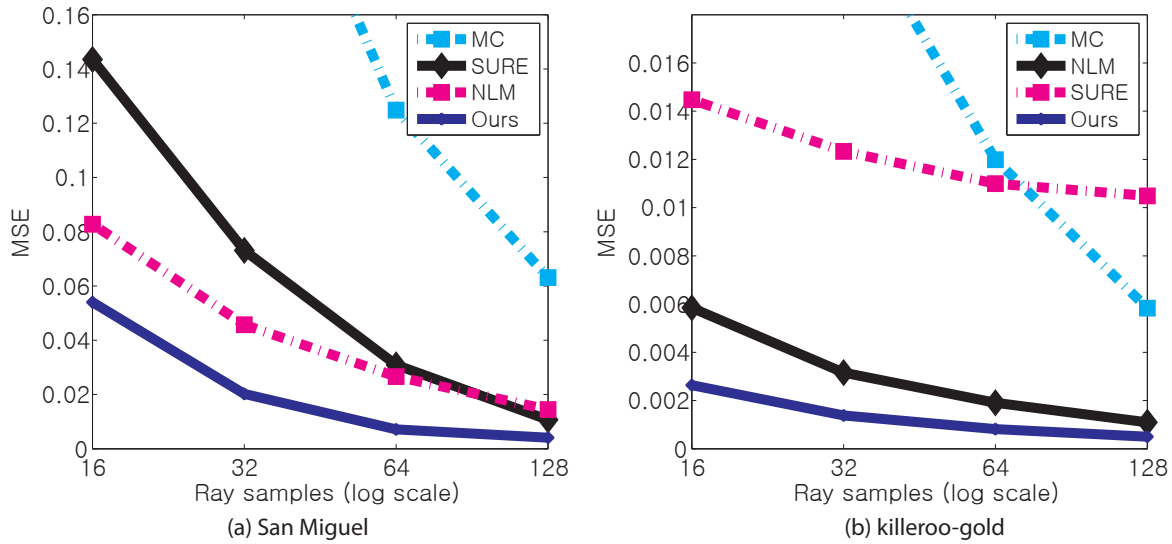


Figure 5.9: These graphs show convergence plots in different methods; the curve of MC sampling is invisible in the left because of its too high MSE. Our method outperforms previous methods across all the tested sample counts. Note that SURE shows even higher MSE than MC uniform sampling beyond 70 spp on the right.

while SURE does.

In Fig. 5.7 we can verify the robustness of tested methods against DOF effects with multiple dragons that have quite different depths. Given the equal-time comparison our method produces the best results in terms of visual quality and MSE. In the defocused area (the second row) NLM and SURE shows under-blurred results on the defocused dragons and textured floor. In the focused area (the first row) NLM and SURE generate under-blurred and over-blurred results, respectively. On the other hand, our method preserves detailed curvatures thanks to the small bandwidth applied to the normal feature. This example clearly indicates that a globally fixed bandwidth for geometry cannot give optimal results. For example, if we use larger bandwidths in SURE, we may remove the noise in the defocused area. It leads, however, to more bias on the focused area. Although SURE considers sample variances of the geometry to reduce noise from distributed effects, it still leaves noticeable noise. On the other hand, our method adaptively removes noisy input features introduced by distribution effects. Our truncated SVD performs this dimension reduction automatically and locally.

The pool scene (Fig. 5.8) with motion blur effects is a challenging benchmark for filtering methods guided by the geometry. For example, the motion of the orange ball (the first row) makes texture buffers noisy, and the fastest motion of the white ball (the second row) introduces very noisy geometry buffers in all the features. Another challenge in this scene is spike noises that still remain even when 16K spp for the reference are generated. NLM generates a relatively good image on the moving orange ball, but leaves noticeable artifacts on the white ball. SURE shows under-blurred results on the both balls because of highly noisy geometry buffers. Our method shows visually better and produces the lowest MSEs among all the tested methods.

Comparison with RPF. Although RPF [63] is not suitable for adaptive rendering because of its slow performance and lack of error analysis, it considers the relative importance of feature types like our approach, and thus we compare ours against to RPF. Fig. 5.10 shows quality comparisons with the dof-

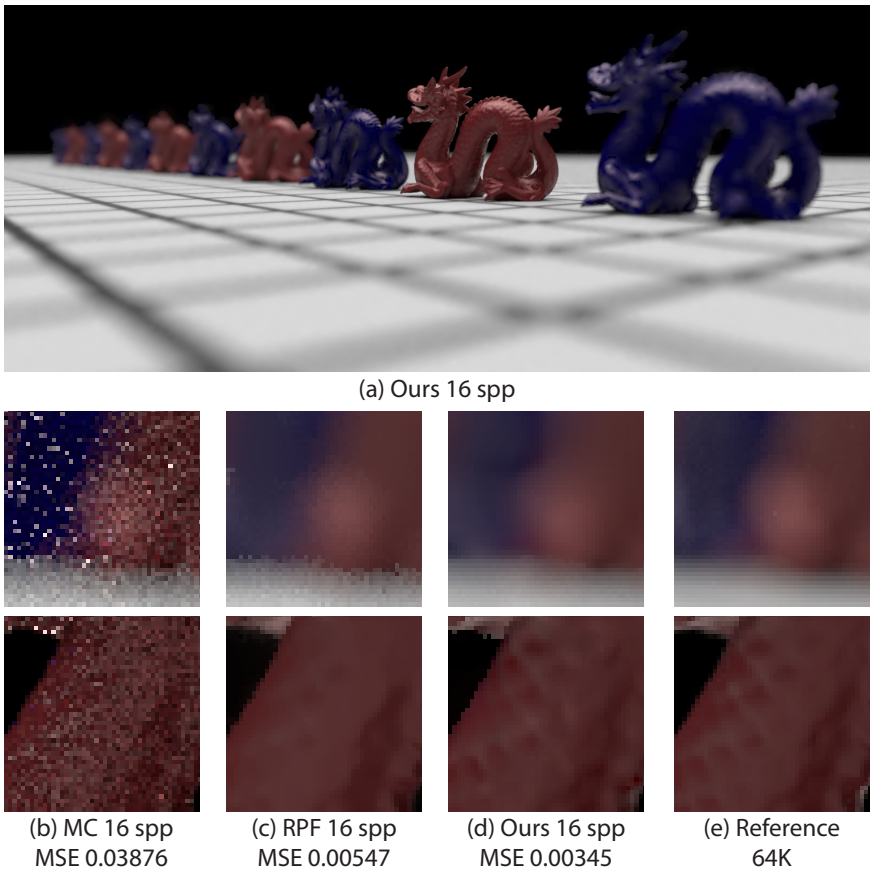
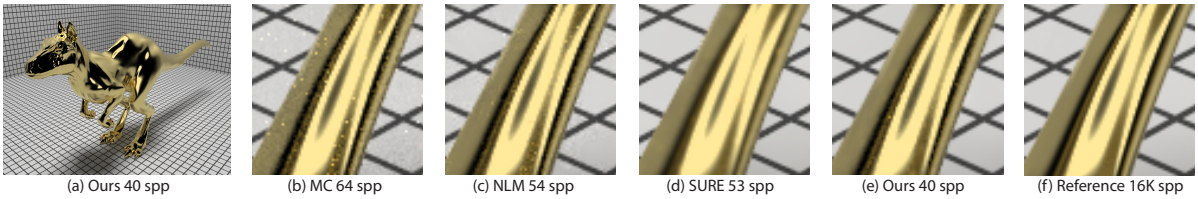


Figure 5.10: Comparison with random parameter filtering (RPF) in the dof-dragon. Our method shows more visually pleasing and numerically better results over RPF. Similar results are observed for other benchmarks.



(a) Ours 40 spp (193 s) MSE 0.00116 | (b) MC 64 spp (196 s) MSE 0.01199 | (c) NLM 54 spp (199 s) MSE 0.00215 | (d) SURE 53 spp (202 s) MSE 0.01148 | (e) Ours 40 spp (193 s) MSE 0.00116 | (f) Reference 16K spp

Figure 5.11: Equal-time comparisons in the killeroo-gold scene. The zoom-in inset visualizes highlights on the leg. NLM (c) leaves noise on the leg. SURE (d) and our method (e) show visually pleasing results, but SURE gives over-blurred results. Overall our method shows 10:1 and 2:1 MSE reduction ratios over SURE and NLM, respectively.

dragon. RPF unfortunately leaves noises in some regions, while over-blurring features in other regions. On the other hand, our method produce a better denoising result that is comparable to the reference image computed with 64 k spp. This difference is mainly because our method robustly measures feature bandwidths to minimize MSE. In addition, even a CPU version of our method runs faster by one order of magnitude than RPF.

Chapter 6. Discussions

Multi-resolution subdivision techniques. We have demonstrated our acceleration methods with complex benchmarks. Some of the tested scanned models (e.g., St. Matthew, Dragon, etc.) are highly tessellated. One can simplify these highly tessellated models and render them by using multi-resolution subdivision techniques [7, 71, 16] without significant image quality degradations. Our ray reordering can further improve the performance of these multi-resolution methods. More specifically speaking, even though multi-resolution approaches can reduce the number of triangles that have to be processed, we usually have to deal with a huge number of subdivided triangles to provide high quality rendering. Therefore, typical multi-resolution approaches rely heavily on smart caching schemes. Our ray reordering method can maximize the benefits of these kinds of caching schemes by improving the ray coherence. Therefore, our method is not competing with the multi-resolution methods, but complementary to each other.

Optimized ray tracers. In the experimental validation of our ray reordering, we have used a slow ray tracer which does not have a high processing throughput. However, the performance improvement caused by reducing the number of cache misses can be higher with more highly optimized ray tracing systems. This is because in these optimized systems, the data access time caused by cache misses will take relatively higher portions in the overall ray tracing time. Therefore, our reordering method that reduces this data access time can achieve higher improvements with more optimized ray tracers. To support this argument, we implement multi-BVHs [20] that provide an efficient SIMD-based triangle packet and thus improve the ray processing throughput, especially for incoherent secondary rays. We choose to have four child nodes for an intermediate node and four triangles in the leaf node of the multi-BVH. Also, in the 32 bit machine, only 39% of the whole data of the Sponza scene is cached in main memory. Given the multi-BVH, we use a single-ray and single-triangle traversal by disabling to use the SIMD-based triangle packet, as a lower performance ray tracer. In this ray tracer, the ray processing throughputs is improved from 5.49 K rays per second (RPS) to 27.4 K RPS, resulting in a 4.99 times improvement, by reordering rays. Then, we enable the SIMD-based triangle packet traversal for a higher performance ray tracer. In this case, by reordering rays, the ray processing throughput is improved from 5.5 K RPS to 28.92 K RPS, leading to a 5.26 times improvement, a higher improvement than that with the lower performance ray tracer.

Non-local means filtering. We have implemented our virtual flash image based denoising on top of non-local means filtering, which has been known to be better than bilateral filtering for denoising photographs [5]. Because non-local means filtering computes filtering weights based on patches, it can be more robust against noise than (cross) bilateral filtering that adopts the pixel-wise weight computation (see Fig. 4.6). Even though the noise in virtual flash images is much reduced compared to that of the input noise images, the virtual flash images can still have a high variance for some scenes, especially when scenes have complex illumination configurations. One example includes the glossy metal trash can, as shown in Fig. 4.13. Even in this case, non-local means filtering shows better denoising results than cross bilateral filtering.

Cross bilateral filtering. Our virtual flash image can be naturally used with cross bilateral filtering

since the flash image is an edge-stopping function and independent from a specific filtering framework, although we have demonstrated that our method with non-local means filtering. In rendering, the cross bilateral has been widely applied when we denoise rendering noise using some rendering-specific features since the features are easily considered as edge-stopping functions in the filtering framework [41]. One of the challenging issues is that filtering bandwidths for included features should be properly selected since it controls the bias-variance tradeoff of filtering. Unfortunately, the automatic bandwidth selection based on error analysis in the cross bilateral filter remains an unresolved problem. For example, Li et al. [41] proposed a bandwidth selection for the spatial kernel, but other bandwidths for geometric features are set manually. As demonstrated by the random parameter filtering [**?**], optimal bandwidths for other features should be estimated locally to produce a high quality reconstruction result. Our optimal bandwidth estimation addresses the mentioned problem within the local regression framework.

Pixel-based local regression. Our adaptive rendering work has been implemented as pixel-based filtering that is commonly used in image filtering methods [59, 60, 41]. Our regression method can be performed directly with samples and it can potentially give better results. In practice working directly with samples requires a high memory and performance overhead, because all the samples should be stored. Instead, we store only the mean and variance for each feature type per pixel. We use those values stored at pixels as our samples under our filtering window. As a result, its required memory and performance overhead are independent from the number of ray samples. When we use the GPU to perform our reconstruction method on a 1k by 1k image, it takes 6 s of processing time. The main bottleneck of our method is in its various matrix operations. For complex benchmarks such as the San Miguel scene, it takes approximately 880 s to generate 128 spp. As a result, the overhead of our reconstruction method takes a small portion compared to the overall rendering process, and our method produces visually pleasing results much more efficiently than generating more samples, because of its effectiveness for distributing samples in high error regions. Computation overheads of different methods vary depending on the benchmark on the test, but given the setting described in this paper our reconstruction method is approximately 20% and 40% slower than SURE and NLM, respectively. Nonetheless, its robust error estimation and better reconstruction ability of our method results in higher efficiency in terms of reducing errors.

Animations. Our proposed filtering methods (i.e., virtual flash image based denoising and reconstruction using local regression) can be naturally extended to handle animated images. As a straightforward approach, we can simply apply our methods to its individual frame. However, it typically leaves low-frequency noise known as flickering. Instead of the approach, we apply our methods as a post-processing. We first generate each frame in animated sequences, and the pixels in the generated frames can be considered as volumetric pixels. Given a filtering window (e.g., $7 \times 7 \times 5$) we apply our filtering methods on the volumetric pixels. Specifically, we apply our local regression based reconstruction with an additional feature, a frame number t. In our implementation, the runtime overhead of the filtering for each frame takes 7 s for the San Miguel benchmark given a $7 \times 7 \times 5$ filtering window size.

Rendering-specific features. We have demonstrated that our local regression based reconstruction with geometric features such as normals, textures, and depth. However, additional rendering features such as the frame number in animated sequences can be utilized within our framework since our method supports the additional features through automatic bandwidth selection process for the features. For example, our virtual flash image can be considered as a new feature, and the feature can be used with existing features since all the optimal bandwidths for each feature are automatically estimated. It

can significantly reduce the effort for using additional features. Generally speaking, introducing new rendering features to predict ground truth images is a challenging, but important issue since it typically gives a hint for discerning edges from noise. We leave the exploration of new features as a future work.

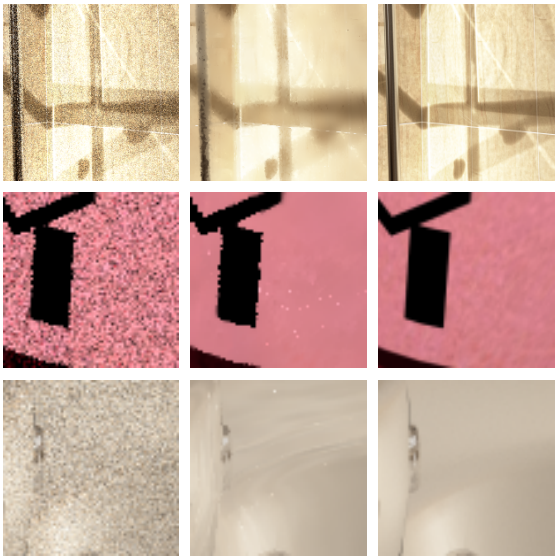
Outliers. In rendering, outliers are often defined as spike noise with extremely high energy. If outliers are not addressed well sampling budgets are drained, because pixels with outliers typically have a very large variance [59]. Furthermore, a reconstruction process requires a larger window size to spread out its energy [60]. In our regression based reconstruction, we use a simple heuristic to suppress its influence, by dividing the computed weights of samples by the variance of the sample mean. This approach works with our local regression approach and has a negligible overhead. Nonetheless we have observed that this simple trick produces visually pleasing results. This is mainly because our method robustly measures bias even with outliers and our reconstruction method restores energy loss of outliers, as the number of samples is increased. Note that high bias values for pixels with outliers cause those pixels to receive appropriate samples and thus the variance of the sample mean tends to decrease. As a result, our reconstruction method puts higher weights on outliers, reducing the energy loss.

Applications to other problems. At a high level our adaptive reconstruction technique for rendering can be applied to many related problems that also rely on G-buffers. An example application is upsampling for rendering. Typically upsampling is performed based on joint bilateral filtering [58] that considers discontinuity based on the G-buffer. As we pointed out before, when G-buffers contain noisy features, it can produce sub-optimal results. Our local regression based reconstructions can address this problem, as demonstrated by denoising process in this paper. Furthermore most prior upsampling methods used in rendering require a uniform sampling of its input image. Our method does not require this restriction and can support irregular sampling.

6.1 Limitations

Our methods have certain limitations. Our reordering method inherits drawbacks of existing reordering methods: to use our reordering method in a ray tracer, the ray tracer should be decomposed into separate ray generation and processing modules, and processing rays are performed iteratively by using these modules. Also, our ray reordering method like other ray reordering methods may not work with shaders that do not allow deferred shading, though most general shaders work with the deferred shading. Although we have demonstrated performance improvements over other cache-oblivious reordering methods that use ray origins or ray directions with all the tested benchmarks, there is no guarantee that our method will improve the performance of ray tracing because of the overhead of our method. Also, our reordering method requires simplified representations of original models. Computing such simplified representations require extra implementation efforts. Moreover, for certain class of models such as forest scenes and furry models, computing high-quality simplified representations may be very difficult, thus lowering the performance benefits of our method in such scenes. Also, as demonstrated in the St. Matthew scene, our method can show a lower performance than optimized cache-aware ray reordering methods.

We use approximate confidence intervals to preserve those edges that a virtual flash image does not contain. The approximate intervals are based on the variances of ray samples generated from Monte Carlo ray tracing, and thus the approximation quality depends on the number of ray samples, according to the central limit theorem. As a result, the computed interval may not be accurate, especially with relatively small numbers of ray samples. For example, the intervals computed from a small ray count



(a) Input image (b) Ours (c) Reference

Figure 6.1: Failure cases of our method with four (first and second row) and sixteen ray samples per pixel (third row). Our method fails to preserve high-frequency edges in caustics of the shower booth scene (first row), and leaves noisy pixels in the toaster scene (second row). Our method also shows under-smoothing artifacts in the highly noisy bathtub of the bathroom scene (third row).

(e.g., 4) can be too wide and fail to preserve high-frequency edges, as demonstrated in the first row of Fig. 6.1. Furthermore, the intervals can be too narrow and thus fail to smooth out noise, as illustrated by the second row of Fig. 6.1; note that our denoised result exhibits noisy pixels. Also, in the highly noisy region (the third row of Fig. 6.1), an insufficient number of homogeneous pixels may be selected because our method does not make use of neighboring pixels outside the confidence intervals. This binary decision makes the stochastic error bounds possible, but can lead to the generation of under-smoothing artifacts. In addition, the virtual flash image can only capture limited subsets of all possible high-frequency edges. Global illumination effects in complex scene configurations, such as glossy-dominant scenes or scenes with participating media, may not be preserved well, especially with low ray samples. The virtual flash image with considering the virtual flash light may include new illumination effects (e.g., specular highlights), which may leave some noise, especially with low ray samples.

In the conference scene (Fig 6.2), path tracing generates strong spike noise. We observed that spike noise is still noticeable even with uniformly sampled 16K rays per pixel. All the tested adaptive rendering methods do not provide satisfactory results because of the frequent presence of spike noise. NLM [60] tends to leave the spike noise on some areas, and SURE [41] generates artifacts (the first row) or remove shadows (the second row). Our method is visually better than the other methods, but still exhibits low-frequency noise. We may further lessen the problem by using a larger window size [60], but this approach often introduces more bias in other scenes. Efficiently addressing outliers without energy loss (i.e. bias) still remains a challenging problem in adaptive rendering techniques. In addition, our error analysis is derived by assuming that input samples satisfy the independence property. Since we use adaptive sampling that depends on the prior sampling pattern, and low discrepancy sampling, our error analysis does not reflect practice in a perfect manner. Nonetheless, our method shows visually pleasing and numerally better results over the state-of-the-art adaptive methods.

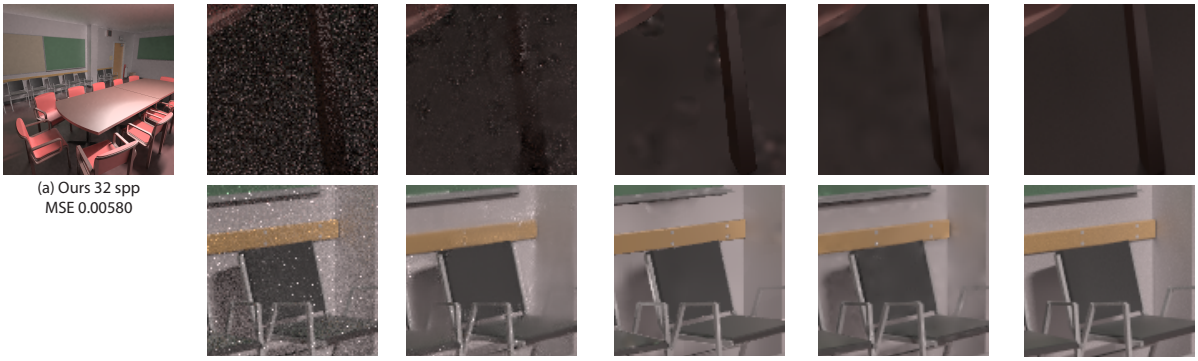


Figure 6.2: Failure case of the conference room. This scene contains a large number of outliers, i.e. spike noise. All the tested methods including ours do not show satisfactory results in terms of quality and MSE. NLM (c) leaves many high frequency noise out, and SURE (d) makes noticeable artifacts in the first row or removes shadows in the second row. Our method (e) shows visually better results than previous methods, but still leaves low frequency noise.

Chapter 7. Conclusion and Future Work

In this Ph.D. thesis, I have proposed three acceleration techniques for improving performance of Monte Carlo (MC) ray tracing. We have presented a cache-oblivious ray reordering method that achieves the performance improvement for various models. Our method reorders rays by computing approximate hit points and efficiently sorting them with the Z-curve. We have showed that our framework has a high modularity with Monte Carlo ray tracing such as path tracing and photon mapping, both of which require lots of incoherent rays to generate realistic visual images.

We have also proposed an efficient and robust image denoising method for noisy images generated by Monte Carlo ray tracing in order to reduce the number of ray samples required for producing a high-quality rendered images. Our filtering method achieves high-quality denoising results based on the novel concept of virtual flash images. These were motivated by the flash/non-flash image enhancement in computational photography, and we have presented the necessary modifications for denoising rendered images. Virtual flash images provide a novel way to capture image features based on actual shaded results. In addition to the virtual flash images, we have introduced the idea of using homogeneous pixels during the denoising process. Considering only homogeneous pixels makes our denoising method robust to inappropriate parameter values and provides a provable stochastic error bound. This alleviates the excessive trial and error adjustment of user-defined parameters.

We have presented a novel, weighted local regression based adaptive rendering technique for efficiently and robustly handling a wide variety of rendering effects. We locally identify noisy features based on our truncated SVD and perform our reconstruction with a reduced feature space. We then parameterize the bias and variance of our reconstruction error and find the optimal bandwidths for feature types. We also adopt an iterative sampling process that distributes the available ray budgets to regions with high errors according to our error metric.

We demonstrated robustness and efficiency of our proposed acceleration methods with a diverse set of complex benchmarks models under different rendering effects, and showed that our method produces a robust and consistent improvement over the state-of-the-art techniques. By reordering these rays based on the hit point heuristic, we have achieved significant performance improvements over other simple cache-oblivious ray reordering methods that use ray origins or ray directions for massive models. Moreover, our ray reordering method shows a performance improvement for small models that can fit into main memory. This performance improvement is caused by reducing the cache misses of the L1/L2 caches. Also, our method shows comparable performances even with the optimized cache-aware ray reordering method proposed by Pharr el al. [52]. By applying the virtual flash image and homogeneous pixels, our filtering method achieves better denoising results in terms of visual quality and numerical accuracy given a wide set of benchmarks. Furthermore, we have demonstrated that our method works well on a wider range of scene configurations than existing denoising methods. Finally, our adaptive rendering method shows visually and numerically better rendering results compared to the results of the state-of-the-art approaches such as SURE [41] and NLM [60].

7.1 Future Work

There are many exciting future research directions lying ahead. Currently, we have tested our ray reordering method only with the CPU architecture. It will be very interesting to see how our reordering method can be extended to handle incoherent rays and improve GPU cache utilizations in GPU-based global illumination methods. It will be also interesting to apply our method to hybrid ray tracers that run on both CPUs and GPUs. Furthermore, although we have presented a cache-oblivious computational reordering method for ray tracing, its idea can be applied to many different applications whose main bottleneck is in the data access time. Therefore, we would like to extend our current reordering to different computer graphics applications.

There are also rooms for improvement on the virtual flash images and homogeneous pixels. One direction would be to extend virtual flash images for the rendering of participating media. This is a challenging problem, because even the computation of single scattering, which roughly corresponds to direct illumination in virtual flash images, requires many samples. In addition, we would like to further reduce the variance of virtual flash images through the rejection of unnecessary ray paths generated from glossy materials. For example, it would be more desirable to reject some ray paths generated from glossy materials that may not lead to high-frequency edges. In order to handle participating media and general glossy materials efficiently, we need to find an appropriate subset of light paths that is inexpensive to compute but captures almost all high-frequency features. We would also like to refine the estimation of confidence intervals. For example, combining confidence intervals with a functional relationship between samples [63] would be an interesting research direction. It is also interesting to investigate how virtual flash images and homogeneous pixels can improve other image-based denoising methods.

One of the most difficult challenges in Monte Carlo ray tracing is to robustly handle outliers. Outliers in Monte Carlo rendering pose a unique problem that does not appear in statistics and image processing field, since those outliers are not noise, but are meaningful signal. We plan to investigate a number of different approaches (e.g., spreading out the loss energy into the reconstructed function) within our adaptive rendering framework in a robust way. Another interesting question is to investigate fundamental differences and efficiency between our local regression based image-space adaptive method and photon mapping, since the density estimation and our local regression stem from the same statistical assumptions. One fundamental difference is that our method optimizes filtering bandwidths in a *data-driven* way by considering sample pairs of $\mathbf{x}^i$ and y^i, in addition to its efficiency due to the nature of the image-space approach, while photon mapping focuses more on asymptotic reduction rates. We believe that studying these relationships will broaden our understanding on these approaches and enable us to design a better approach. Finally in the current approach we have found that the local linear model works better over other higher orders. A neglected aspect is considering the use of a constant model in place of the local linear one. Using the constant model can be considered as counter-intuitive, but can be even more efficient and robust than the local linear model. In addition, we would like to extend our image-space adaptive rendering into an efficient multi-dimensional adaptive rendering so that an adaptive sample allocation on other random parameter spaces such as lens can be performed. The samples in Monte Carlo ray tracing are in a high dimensional space, but we may approximate them by using low dimensional structures locally in order to alleviate the curse of dimensionality, while minimizing the loss of information.

Lastly, developing acceleration techniques for interactive or real-time ray tracing would be also interesting research directions. I would like to design a progressive image filtering method that takes samples instead of pixels as input, since we can miss small details when averaging all the samples at a

pixel. The straightforward approach is to store all the samples, but the overhead in terms of storage and performance would be high especially with a large sample count. Furthermore, analyzing errors of noisy Monte Carlo ray traced images with very small ray counts (e.g., four per pixel) would be challenging since estimated errors (e.g., variance) can be severely under- or over-estimated. However, it can be important for improving the quality of interactive rendering that uses a limited ray budget.

References

[1] A. Adams, J. Baek, and M. Davis, "Fast high-dimensional filtering using the permutohedral lattice," *Computer Graphics Forum*, vol. 29, no. 2, pp. 753–762, 2010.

[2] L. Arge, G. S. Brodal, and R. Fagerberg, *Cache-Oblivious Data Structures in Handbook of Data Structures*. CRC Press, 2005.

[3] P. Bauszat, M. Eisemann, and M. Magnor, "Guided image filtering for interactive high-quality global illumination," *Computer Graphics Forum*, vol. 30, no. 4, pp. 1361–1368, 2011.

[4] S. Boulos, I. Wald, and C. Benthin, "Adaptive ray packet reordering," in *IEEE Symp. on Interactive Ray Tracing*, Aug. 2008, pp. 131–138.

[5] A. Buades, B. Coll, and J. M. Morel, "A review of image denoising algorithms, with a new one," *Multiscale Model. Simul.*, vol. 4, no. 2, pp. 490–530, 2005.

[6] B. Budge, T. Bernardin, J. A. Stuart, S. Sengupta, K. I. Joy, and J. D. Owens, "Out-of-core data management for path tracing on hybrid resources," *Comput. Graph. Forum (EG)*, vol. 28, no. 2, pp. 385–396, 2009.

[7] P. H. Christensen, D. M. Laur, J. Fong, W. L. Wooten, and D. Batali, "Ray differentials and multiresolution geometry caching for distribution ray tracing in complex scenes," *Computer Graphics Forum*, vol. 22, no. 3, pp. 543–552, Sep. 2003.

[8] W. S. Cleveland and C. L. Loader, "Smoothing by Local Regression: Principles and Methods," in *Statistical Theory and Computational Aspects of Smoothing*. New York: Springer, 1996, pp. 10–49.

[9] D. Cline, K. Steele, and P. K. Egbert, "Lightweight bounding volumes for ray tracing," *Journal of Graphics Tools*, vol. 11, no. 4, pp. 61–71, 2006.

[10] R. L. Cook, J. Halstead, M. Planck, and D. Ryu, "Stochastic simplification of aggregate detail," *ACM Trans. Graph.*, vol. 26, no. 3, p. 79, 2007.

[11] R. L. Cook, T. Porter, and L. Carpenter, "Distributed ray tracing," *SIGGRAPH Comput. Graph.*, vol. 18, no. 3, pp. 137–145, Jan. 1984. [Online]. Available: http://doi.acm.org/10.1145/964965.808590

[12] L. Dagum and R. Menon, "OpenMP: an industry standard api for shared-memory programming," *IEEE Computational Sci. and Engineering*, vol. 5, pp. 46–55, 1998.

[13] H. Dammertz, D. Sewtz, J. Hanika, and H. P. A. Lensch, "Edge-avoiding A-Trous wavelet transform for fast global illumination filtering," in *High Performance Graphics*, 2010, pp. 67–75.

[14] D. E. DeMarle, C. P. Gribble, and S. G. Parker, "Memory-savvy distributed interactive ray tracing." in *EGPGV*, 2004, pp. 93–100.

[15] P. Diaz-Gutierrez, A. Bhushan, M. Gopi, and R. Pajarola, "Constrained Strip Generation and Management for Efficient Interactive 3D Rendering," in *Computer Graphics International*, 2005, pp. 115–121.

[16] P. Djeu, W. Hunt, R. Wang, I. Elhassan, G. Stoll, and W. R. Mark, "Razor: An architecture for dynamic multiresolution ray tracing," The Univ. of Texas at Austin, Dept. of Computer Sciences, Tech. Rep. TR-07-52, January 24 2007.

[17] D. Donoho and I. M. Johnstone, "Adapting to unknown smoothness via wavelet shrinkage," *Journal of the American Statistical Association*, vol. 90, pp. 1200–1224, 1995.

[18] K. Egan, F. Hecht, F. Durand, and R. Ramamoorthi, "Frequency analysis and sheared filtering for shadow light fields of complex occluders," *ACM Trans. Graph.*, vol. 30, no. 2, pp. 9:1–9:13, 2011.

[19] E. Eisemann and F. Durand, "Flash photography enhancement via intrinsic relighting," in *ACM SIGGRAPH*, 2004, pp. 673–678.

[20] M. Ernst and G. Greiner, "Multi bounding volume hierarchies," in *Interactive Ray Tracing. IEEE Symp. on*, Aug. 2008, pp. 35–40.

[21] M. Frigo, C. Leiserson, H. Prokop, and S. Ramachandran, "Cache-oblivious algorithms," in *Foundations of Computer Science*, 1999, pp. 285–297.

[22] M. Garland and P. Heckbert, "Surface simplification using quadric error metrics," in *SIGGRAPH 97 Proceedings*, 1997, pp. 209–216.

[23] C. M. Goral, K. E. Torrance, D. P. Greenberg, and B. Battaile, "Modelling the interaction of light between diffuse surfaces," in *Computer Graphics (SIGGRAPH '84 Proceedings)*, vol. 18, no. 3, Jul. 1984, pp. 212–22.

[24] C. P. Gribble and K. Ramani, "Coherent ray tracing via stream filtering," in *IEEE Symposium on Interactive Ray Tracing*, 2008, pp. 59–66.

[25] T. Hachisuka, W. Jarosz, and H. W. Jensen, "A progressive error estimation framework for photon density estimation," in *ACM SIGGRAPH Asia*, 2010, pp. 144:1–144:12.

[26] T. Hachisuka, W. Jarosz, R. P. Weistroffer, K. Dale, G. Humphreys, M. Zwicker, and H. W. Jensen, "Multidimensional adaptive sampling and reconstruction for ray tracing," in *ACM SIGGRAPH*, 2008, pp. 33:1–33:10.

[27] P. Hansen, "The truncated svd as a method for regularization," *BIT Numerical Mathematics*, vol. 27, no. 4, pp. 534–553, 1987.

[28] V. Havran, "Cache sensitive representation for the bsp tree," in *Proc. of Compugraphics*, 1997.

[29] A. Hayter, *Probability and statistics for engineers and scientists 3rd.* Brooks/Cole Publishing Co., 2007.

[30] P. S. Heckbert and P. Hanrahan, "Beam tracing polygonal objects," in *SIGGRAPH.* New York, NY, USA: ACM Press, 1984, pp. 119–127.

[31] H. W. Jensen, *Realistic Image Synthesis Using Photon Mapping.* AK Peters, 2005.

[32] H. W. Jensen, S. R. Marschner, M. Levoy, and P. Hanrahan, "A practical model for subsurface light transport," in *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*. New York, NY, USA: ACM, 2001, pp. 511–518.

[33] J. T. Kajiya, "The rendering equation," in *SIGGRAPH '86: Proceedings of the 13th annual conference on Computer graphics and interactive techniques*. New York, NY, USA: ACM, 1986, pp. 143–150.

[34] C. Kervrann and J. Boulanger, "Optimal spatial adaptation for patch-based image denoising," *Image Processing, IEEE Trans. on*, vol. 15, no. 10, pp. 2866–2878, 2006.

[35] T.-J. Kim, Y. Byun, Y. Kim, B. Moon, S. Lee, and S.-E. Yoon, "HCCMeshes: Hierarchical-culling oriented compact meshes," *Computer Graphics Forum (Eurographics)*, vol. 29, no. 2, pp. 299–308, 2010.

[36] T.-J. Kim, B. Moon, D. Kim, and S.-E. Yoon, "RACBVHs: Random-accessible compressed bounding volume hierarchies," *IEEE Trans. on Visualization and Computer Graphics*, vol. 16, no. 2, pp. 273–286, 2010.

[37] C. Lauterbach, M. Garland, S. Sengupta, D. Luebke, and D. Manocha, "Fast bvh construction on gpus," *Computer Graphics Forum (EG)*, vol. 28, no. 2, pp. 375–384, 2009.

[38] C. Lauterbach, S.-E. Yoon, D. Tuft, and D. Manocha, "RT-DEFORM: Interactive ray tracing of dynamic scenes using bvhs," in *IEEE Symp. on Interactive Ray Tracing*, 2006, pp. 39–46.

[39] C. Lauterbach, S.-E. Yoon, M. Tang, and D. Manocha, "ReduceM: Interactive and memory efficient ray tracing of large models," *Comput. Graph. Forum (EGSR)*, vol. 27, no. 4, pp. 1313–1321, 2008.

[40] J. Lehtinen, T. Aila, S. Laine, and F. Durand, "Reconstructing the indirect light field for global illumination," *ACM Trans. Graph.*, vol. 31, no. 4, pp. 51:1–51:10, 2012.

[41] T.-M. Li, Y.-T. Wu, and Y.-Y. Chuang, "Sure-based optimization for adaptive sampling and reconstruction," *ACM Trans. Graph.*, vol. 31, no. 6, pp. 194:1–194:9, Nov. 2012. [Online]. Available: http://doi.acm.org/10.1145/2366145.2366213

[42] C. Loader, "Bandwidth selection: classical or plug-in?" *The Annals of Statistics*, vol. 27, no. 2, pp. 415–438, 1999.

[43] D. Luebke, M. Reddy, J. Cohen, A. Varshney, B. Watson, and R. Huebner, *Level of Detail for 3D Graphics*. Morgan-Kaufmann, 2002.

[44] E. Mansson, J. Munkberg, and T. Akenine-Moller, "Deep coherent ray tracing," in *IEEE Symp. on Interactive Ray Tracing*, 2007, pp. 79–85.

[45] M. D. McCool, "Anisotropic diffusion for Monte Carlo noise reduction," *ACM Trans. Graph.*, vol. 18, no. 2, pp. 171–194, 1999.

[46] D. P. Mitchell, "Generating antialiased images at low sampling densities," in *ACM SIGGRAPH*, vol. 21, 1987, pp. 65–72.

[47] P. Navratil, D. Fussell, C. Lin, and W. Mark, "Dynamic ray scheduling to improve ray coherence and bandwidth utilization," in *IEEE Symposium on Interactive Ray Tracing*, 2007, pp. 95–104.

[48] R. S. Overbeck, C. Donner, and R. Ramamoorthi, "Adaptive wavelet rendering," in *ACM SIGGRAPH Asia 09*, 2009, pp. 140:1–140:12.

[49] C. Öztireli, G. Guennebaud, and M. Gross, "Feature preserving point set surfaces based on non-linear kernel regression," in *Proceedings of Eurographics 2009*, 2009, pp. 493–501.

[50] P. Perona and J. Malik, "Scale-space and edge detection using anisotropic diffusion," *Pattern Analysis and Machine Intel., IEEE Trans. on*, vol. 12, no. 7, pp. 629 –639, 1990.

[51] G. Petschnigg, R. Szeliski, M. Agrawala, M. Cohen, H. Hoppe, and K. Toyama, "Digital photography with flash and no-flash image pairs," *ACM Trans. Graph.*, vol. 23, no. 3, pp. 664–672, 2004.

[52] M. Pharr, C. Kolb, R. Gershbein, and P. Hanrahan, "Rendering complex scenes with memory-coherent ray tracing," in *ACM SIGGRAPH*, 1997, pp. 101–108.

[53] M. Pharr and G. Humphreys, *Physically Based Rendering: From Theory to Implementation*. Morgan Kaufmann Publishers Inc., 2004.

[54] ——, *Physically Based Rendering: From Theory to Implementation 2nd*. Morgan Kaufmann Publishers Inc., 2010.

[55] M. Ramasubramanian, S. N. Pattanaik, and D. P. Greenberg, "A perceptually based physical error metric for realistic image synthesis," in *ACM SIGGRAPH*, 1999, pp. 73–82.

[56] A. Reshetov, "Faster ray packets - triangle intersection through vertex culling," in *IEEE Symp. on Interactive Ray Tracing*, 2007, pp. 105–112.

[57] A. Reshetov, A. Soupikov, and J. Hurley, "Multi-level ray tracing algorithm," *ACM Trans. Graph.*, vol. 24, no. 3, pp. 1176–1185, 2005.

[58] T. Ritschel, T. Engelhardt, T. Grosch, H.-P. Seidel, J. Kautz, and C. Dachsbacher, "Micro-rendering for scalable, parallel final gathering," in *ACM SIGGRAPH Asia*, 2009, pp. 1–8.

[59] F. Rousselle, C. Knaus, and M. Zwicker, "Adaptive sampling and reconstruction using greedy error minimization," in *SIGGRAPH Asia*, 2011, pp. 159:1–159:12.

[60] ——, "Adaptive rendering with non-local means filtering," *ACM Trans. Graph.*, vol. 31, no. 6, pp. 195:1–195:11, 2012.

[61] D. Ruppert and M. Wand, "Multivariate locally weighted least squares regression," *The annals of statistics*, vol. 22, no. 3, pp. 1346–1370, 1994.

[62] H. Sagan, *Space-Filling Curves*. Springer-Verlag, 1994.

[63] P. Sen and S. Darabi, "On filtering the noise from the random parameters in monte carlo rendering," *ACM Trans. Graph.*, vol. 31, no. 3, pp. 18:1–18:15, 2012.

[64] P. Shirley, T. Aila, J. Cohen, E. Enderton, S. Laine, D. Luebke, and M. McGuire, "A local image reconstruction algorithm for stochastic rendering," in *I3D*, 2011, pp. 9–14.

[65] P. Shirley and R. K. Morley, *Realistic Ray Tracing*, 2nd ed. AK Peters, 2003.

[66] C. Silva, Y.-J. Chiang, W. Correa, J. El-Sana, and P. Lindstrom, "Out-of-core algorithms for scientific visualization and computer graphics," in *IEEE Visualization Course Notes*, 2002.

[67] C. Soler, K. Subr, F. Durand, N. Holzschuch, and F. Sillion, "Fourier depth of field," *ACM Trans. Graph.*, vol. 28, no. 2, pp. 18:1–18:12, 2009.

[68] J. Steinhurst, G. Coombe, and A. Lastra, "Reordering for cache conscious photon mapping," in *Graphics Interface*, 2005, pp. 97–104.

[69] A. Stephens, S. Boulos, J. Bigler, I. Wald, and S. G. Parker, "An Application of Scalable Massive Model Interaction using Shared Memory Systems," in *EGPGV*, 2006, pp. 19–26.

[70] G. Stewart, "On the early history of the singular value decomposition," *SIAM review*, vol. 35, no. 4, pp. 551–566, 1993.

[71] E. Tabellion and A. Lamorlette, "An approximate global illumination system for computer generated films," *ACM Trans. Graph.*, vol. 23, no. 3, pp. 469–476, 2004.

[72] H. Takeda, S. Farsiu, and P. Milanfar, "Kernel regression for image processing and reconstruction," *Image Processing, IEEE Transactions on*, vol. 16, no. 2, pp. 349 –366, 2007.

[73] R. Tamstorf and H. W. Jensen, "Adaptive sampling and bias estimation in path tracing," in *Rendering Techniques*, 1997, pp. 285–295.

[74] C. Tomasi and R. Manduchi, "Bilateral filtering for gray and color images," in *ICCV*. IEEE Computer Society, 1998, p. 839.

[75] P. van Emde Boas, "Preserving order in a forest in less than logarithmic time and linear space," *Inf. Process. Lett.*, vol. 6, pp. 80–82, 1977.

[76] J. S. Vitter, "External memory algorithms and data structures: dealing with massive data," *ACM Comput. Surv.*, vol. 33, no. 2, pp. 209–271, 2001.

[77] I. Wald, S. Boulos, and P. Shirley, "Ray Tracing Deformable Scenes using Dynamic Bounding Volume Hierarchies," *ACM Transactions on Graphics*, vol. 26, no. 1, p. 6, 2007.

[78] I. Wald, A. Dietrich, and P. Slusallek, "An Interactive Out-of-Core Rendering Framework for Visualizing Massively Complex Models," in *EG Symp. on Rendering*, 2004, pp. 82–91.

[79] I. Wald, W. R. Mark, J. Günther, S. Boulos, T. Ize, W. Hunt, S. G. Parker, and P. Shirley, "State of the Art in Ray Tracing Animated Scenes," in *Eurographics State of the Art Reports*, 2007.

[80] I. Wald, P. Slusallek, and C. Benthin, "Interactive distributed ray tracing of highly complex models," in *EG Workshop on Rendering*, 2001, pp. 277–288.

[81] T. Whitted, "An improved illumination model for shaded display," *Commun. ACM*, vol. 23, no. 6, pp. 343–349, 1980.

[82] ——, "An improved illumination model for shaded display," *Communications of the ACM*, vol. 23, no. 6, pp. 343–349, Jun. 1980.

[83] R. Xu and S. N. Pattanaik, "A novel Monte Carlo noise reduction operator," *IEEE Comput. Graph. Appl.*, vol. 25, no. 2, pp. 31–35, 2005.

[84] S.-E. Yoon, E. Gobbetti, D. Kasik, and D. Manocha, *Real-Time Massive Model Rendering*. Morgan & Claypool Publisher, 2008.

[85] S.-E. Yoon and P. Lindstrom, "Mesh layouts for block-based caches," *IEEE Trans. on Visualization and Computer Graphics (Proc. Visualization)*, vol. 12, no. 5, pp. 1213–1220, 2006.

[86] S.-E. Yoon, P. Lindstrom, V. Pascucci, and D. Manocha, "Cache-Oblivious Mesh Layouts," *ACM Transactions on Graphics (SIGGRAPH)*, vol. 24, no. 3, pp. 886–893, 2005.

[87] S.-E. Yoon and D. Manocha, "Cache-efficient layouts of bounding volume hierarchies," *Computer Graphics Forum (Eurographics)*, vol. 25, no. 3, pp. 507–516, 2006.

[88] B. Zhang and J. Allebach, "Adaptive bilateral filter for sharpness enhancement and noise removal," *Image Processing, IEEE Trans. on*, vol. 17, no. 5, pp. 664 –678, 2008.

Summary

Acceleration Techniques for Monte Carlo Ray Tracing

몬테카를로 광선 추적법 (Monte Carlo ray tracing)은 실사 렌더링 효과를 시뮬레이션하기 위해 쓰이는 효과적인 기술로 알려져 있다. 그러나 몬테카를로 광선 추적법은 실사와 같은 렌더링 효과를 얻기 위해서 많은 광선 샘플을 이용해야 하는 단점을 가지고 있다. 이는 몬테카를로 광선 추적법의 렌더링 성능 저하를 일으키는 근본적인 문제이다. 본 졸업 논문에서는 몬테카를로 광선 추적법의 성능 개선이라는 도전적인 문제를 풀고자 하며 이를 위해서 새로운 가속화 방법 세가지를 제안한다. 우선 광선 재정렬 방법을 통해 광선들이 3차원 공간상에서 처리될 때 발생하는 캐시 미스 횟수를 줄이기 위해 메모리 접근의 지역성을 높이기 위한 방법을 제안한다. 이를 위해 교차점 휴리스틱 방법에 기반한 캐시 친하적 광선 재정렬 방법을 제시하였다. 몬테카를로 광선 추적법의 성능을 더욱 개선하기 위해서는 잡음 없는 선명한 이미지를 얻기 위해 요구되는 광선 샘플 수를 줄이는 일이 필수적이며 이를 위해 몬테카를로 광선 추적법에 의해 생성되는 이미지의 잡음 제거를 위한 필터링 방법을 제안한다. 효과적 이미지 필터링을 위해서는 이미지의 에지와 잡음을 구별하여 에지는 보존하고 잡음만을 제거해야 하는데 이를 위해 새로운 에지 함수인 가상 플래시 이미지를 제안하였다. 가상 플래시 이미지는 다양한 렌더링 효과에 의해 생성되는 에지를 포함하고 있으며, 우리는 이 새로운 이미지를 통해 적은 광선 샘플 수만 이용해서 에지를 보존하면서 렌더링 이미지의 잡음을 효과적으로 제거하였다. 마지막으로 이미지 필터링 기술을 더욱 효과적으로 적용하기 위해서 이미지 필터링 오류를 추정하여 오류를 최소화하도록 이미지의 각 픽셀마다 최적의 필터 너비를 사용하도록 로컬 리그레션 (local regression)이론에 기반한 새로운 이미지 기반 적응형 렌더링 기술을 제시하였다. 또한 픽셀마다 오류를 추정하여 최적의 광선 샘플수를 사용하여 선명한 렌더링 이미지를 얻기 위해 요구되는 광선 샘플 수를 줄였다. 본 박사학위 논문에서 제안한 최적화 방법들이 기존의 최신 방법들 대비 몬테카를로 광선 추적법의 성능을 크게 향상시킴을 다양한 벤치마크를 이용하여 검증하였다.

핵심어: 몬테카를로 광선 추적법, 광선 재배열, 이미지 필터링, 적응형 렌더링

감 사 의 글

학위기간동안 좋은 연구를 할 수 있도록 도와주시고, 좋은 연구자가 되기 위한 덕목을 가르쳐주신 윤성의 교수님께 제일 먼저 고맙다는 말을 하고 싶습니다. 또한 학위논문 작성을 잘 마무리 할 수 있도록 조언을 해주신 심사위원 (김경국, 김민혁, 박진아, 이성길 교수님) 분들께 감사의 말을 전하고 싶습니다. 논문 제출 때 다양한 코멘트를 주며 학위과정을 잘 마무리 할 수 있도록 옆에서 힘이 되어준 SGLAB 동료 여러분 감사합니다. 학위 과정을 비슷한 시기에 시작하여 흥미로운 연구 주제들에 대해 활발한 토론을 해준 김태준, 김덕수, 이정환, 허재필, Pio Claudio 에게 특별히 감사의 인사를 하고 싶습니다. 마지막으로 옆에서 늘 힘이 되어준 아내 보라에게 고맙다는 말을 하고 싶습니다.

Curriculum Vitae

Name : Moon, Bochang

E-mail : moonbochang@gmail.com

Educations

2010. 2. – 2014. 6. Computer Science at KAIST, Korea (Ph.D.)

2008. 2. – 2010. 1. Computer Science at KAIST, Korea (M.S.)

2004. 3. – 2008. 2. Computer Science at Chung-Ang University, Korea (B.S.)

Career

2011. 6. – 2011. 9. Reserach Intern at Adobe, USA

2009. 2. – 2011. 1. Teaching Assistant at Dept. of Computer Science, KAIST

2008. 2. – 2014. 6. Research Assistant at Scalable Graphics/Geometric Algorithm Lab., KAIST

Publications

1. **Bochang Moon**, Nathan Carr, Sung-Eui Yoon, *Adaptive Rendering based on Weighted Local Regression*, Accepted in ACM Transactions on Graphics, 2014.
2. Hyosub Park, **Bochang Moon**, Soomin Kim, Sung-Eui Yoon, *P-RPF: Pixel-based Random Parameter Filtering for Monte Carlo Rendering*, CAD/Graphics 2013.
3. **Bochang Moon**, Jong Yun Jun, JongHyeob Lee, Kunho Kim, Toshiya Hachisuka, Sung-Eui Yoon, *Robust Image Denoising using a Virtual Flash Image for Monte Carlo Ray Tracing*, Computer Graphics Forum, Vol. 32, number 1, pp. 139-151, 2013.
4. **Bochang Moon**, Youngyong Byun, Tae-Joon Kim, Pio Claudio, Hye-sun Kim, Yun-ji Ban, Seung Woo Nam, Sung-Eui Yoon, *Cache-Oblivious Ray Reordering*, ACM Transactions on Graphics, Vol. 29, No. 3, 2010 (Presented at ACM SIGGRAPH 2011).
5. Tae-Joon Kim, Yongyoung Byun, Yongjin Kim, **Bochang Moon**, Seungyong Lee, Sung-Eui Yoon, *HCCMeshes: Hierarchical-Culling oriented Compact Meshes*, Computer Graphics Forum, vol. 29, no. 2, pp. 299-308, 2010 (Proc. of Eurographics 2010).
6. Tae-Joon Kim, **Bochang Moon**, Duksu Kim, Sung-Eui Yoon, *RACBVHs: Random-Accessible Compressed Bounding Volume Hierarchies*, IEEE Transactions on Visualization and Computer Graphics, vol. 16, no. 2, pp. 273-286, 2010.