The Visual Computer (2018) 34:765–778
https://doi.org/10.1007/s00371-018-1521-y

ORIGINAL ARTICLE

# A framework for developing and benchmarking sampling and denoising algorithms for Monte Carlo rendering

Jonas Deyson Brito dos Santos[1] · Pradeep Sen[2] · Manuel M. Oliveira[1]

Published online: 2 May 2018

**Abstract**
Although many adaptive sampling and reconstruction techniques for Monte Carlo (MC) rendering have been proposed in the last few years, the case for which one should be used for a specific scene is still to be made. Moreover, developing a new technique has required selecting a particular rendering system, which makes the technique tightly coupled to the chosen renderer and limits the amount of scenes it can be tested on to those available for that renderer. In this paper, we propose a renderer-agnostic framework for testing and benchmarking sampling and denoising techniques for MC rendering, which allows an algorithm to be easily deployed to multiple rendering systems and tested on a wide variety of scenes. Our system achieves this by decoupling the techniques from the rendering systems, hiding the renderer details behind an API. This improves productivity and allows for direct comparisons among techniques originally developed for different rendering systems. We demonstrate the effectiveness of our API by using it to instrument four rendering systems and then using them to benchmark several state-of-the-art MC denoising techniques and sampling strategies.

**Keywords** Monte Carlo rendering · Adaptive sampling and Reconstruction · Denoising · Benchmarking

## 1 Introduction

Rendering is one of the most important problems in computer graphics and has been the subject of over half a century of research. In particular, there has been a tremendous amount of exploration on Monte Carlo (MC) physically based rendering systems [10] such as path-tracing [17] and its various extensions [41]. To address shortcomings in Monte Carlo rendering, more than three decades of research has explored a wide variety of different ideas, including adaptive sampling and reconstruction algorithms [14], faster acceleration structures and intersection algorithms [35], improved sampling patterns [15], and Monte Carlo denoisers [18,33,39], to name a few broad categories.

Although developing a new algorithm that successfully improves Monte Carlo rendering in some way is a challenging task in itself, researchers face two further challenges. First, *they must integrate their algorithm into an actual rendering system* so they can test it on complex scenes. After all, renderers have several key components required to produce high-quality images (e.g., scene I/O, samplers, ray-traversal acceleration data structures, primitive-ray intersectors, shading systems, and reconstruction filters), and many of these components are often orthogonal to the algorithm being explored. Therefore, rendering researchers often leverage the infrastructure provided by existing rendering systems. However, integrating a new algorithm into a rendering system is often a time-consuming task, precluding its deployment on multiple renderers to properly test the technique.

Second, *researchers must find several high-quality scenes to test their algorithm and demonstrate their performance*. Since most rendering researchers are not digital artists, constructing complex aesthetically pleasing scenes is often a non-trivial, time-consuming task, and "programmer-art" scenes do not tend to be of the same quality as those constructed by professional artists. Moreover, rendering systems

This work was sponsored by CAPES and CNPq-Brazil (fellowships and Grants 306196/2014-0 and 423673/2016-5), as well as US National Science Foundation Grants IIS-1321168 and IIS-1619376.

✉ Jonas Deyson Brito dos Santos
jdbsantos@inf.ufrgs.br

Pradeep Sen
psen@ece.ucsb.edu

Manuel M. Oliveira
oliveira@inf.ufrgs.br

1 Instituto de Informática, UFRGS, Porto Alegre, RS, Brazil

2 University of California, Santa Barbara, USA

tend to adopt proprietary scene-description formats. Thus, researchers tend to stick to a handful of test scenes available for the selected rendering system.

The consequence of these two challenges is that most researchers are often only able to demonstrate their algorithm on a single rendering system using a small number of test scenes. This significantly limits their ability to thoroughly test and explore the proposed method, and for reviewers to properly evaluate its performance. Furthermore, it is often difficult to compare against existing methods, since they often have been implemented in other rendering systems and tested on different scenes. Having good "apples-to-apples" comparisons is important when trying to gauge the benefits of a new method.

Finally, porting a recently published algorithm to a new rendering system is not easy, since the developers performing the port are usually not experts on the new algorithm, even though they may be very familiar with the target rendering system. Therefore, they usually have to *translate* the available implementation (or the algorithm described in the paper) to the new rendering system. This can introduce bugs in the process and may not produce ideal results, since the algorithmic parameters that worked successfully for one rendering system might not work for the new one. Trying to determine the optimal parameters for an algorithm that one did not develop can be a very time-consuming task.

To address these problems, *we present a novel framework that allows researchers to develop, test, compare, and even deploy sampling and denoising algorithms for Monte Carlo rendering*. Specifically, we propose an application program interface (API) that allows developers to easily port their algorithms to different rendering systems by providing the necessary communication between such algorithms and the other components of an existing rendering system. In other words, instead of having the researchers port their algorithms to multiple rendering systems, we have done the leg work for them by instrumenting rendering systems to provide the necessary services through our API.

Therefore, with our framework a researcher only needs to implement an algorithm once, and can immediately use it with all rendering systems that support our framework. This allows researchers to rapidly test and deploy their algorithms on a range of rendering systems, and test them on a wide variety of scenes. This also enables automatic, independent benchmarking of algorithms, which is quite useful when submitting new techniques for publication.

As a proof of concept, we have initially instrumented four rendering systems (PBRT-v3, PBRT-v2, Mitsuba, and a procedural renderer) with our API, as well as implemented many MC denoising algorithms and three sampling techniques. We plan to open-source our framework so that other Monte Carlo renderers can support the API directly themselves. This will also allow third-party rendering systems to rapidly adopt recently proposed algorithms that conform to our API.

To demonstrate the effectiveness of our framework, we conduct a case study involving Monte Carlo (MC) denoising algorithms. Such a study illustrates key aspects of our system: (1) to provide easy integration of algorithms and rendering systems (by means of just a few calls to the API); (2) to provide an independent benchmark for MC techniques that works across various rendering systems; and (3) to allow developers to evaluate the performance of rendering systems with various algorithms, and vice versa. These are desirable features for algorithm and rendering system developers, as well as for the academic, industry, and end-user communities, who should be able to make better informed decisions when choosing a technique and/or a rendering system to render a given scene.

For our study, we have instrumented many state-of-the-art MC denoising algorithms (e.g., NFOR [7], LBF [18], RHF [11], LWR [23], RDFC [33], RPF [39], SBF [21], NLM [32], and GEM [31]), allowing them to be used with the four rendering systems, even though most of these algorithms have originally been developed for a single renderer. Furthermore, our system's ability to automatically generate benchmark reports allows for the comparison of the different methods on an even playing field. In our study, we compare the performance of different MC denoising methods and discuss some of their identified potential limitations.

Although this paper does not propose a new MC rendering algorithm per se, this kind of *meta-research* system (i.e., a system designed to aid the research process) is not new to the graphics and vision communities. Successful examples include the Middlebury benchmark [36,37], which has transformed the way two-frame dense stereo correspondence algorithms are developed and compared, as well as the benchmarks on Alpha Matting [29,30], optical flow [2,3,5], and video matting [12,13]. More recently, Anderson et al. [1] proposed a framework to compile PDF sampling patterns for Monte Carlo.

Inspired by these works, our system provides test scenes intended to stress the limits of Monte Carlo techniques and reveal their potential limitations. It is extensible, allowing for easy support of new rendering systems, as well as sampling and denoising strategies. The community should be able to contribute new scenes and techniques in a simple way. Our system is publicly available through our project website, providing valuable feedback to the research and industry communities.

In summary, the **contributions** of our work include:

– A framework for developing, testing, and benchmarking sampling and denoising Monte Carlo algorithms (Sect. 3). Our framework decouples the algorithms from rendering systems by means of an API, allowing

researchers to implement their techniques once and run them on any rendering system supporting our framework. It easily incorporates new algorithms, rendering systems, and testing datasets;
- An automatic and independent benchmarking system for comparing Monte Carlo algorithms across multiple rendering systems and supporting a large number of scenes (Sect. 3). This should be a useful tool for assessing the quality of new Monte Carlo algorithms against established ones, especially for submission purposes;
- A detailed evaluation of the state-of-the-art Monte Carlo denoising algorithms using our framework and a discussion of their performance and limitations (Sect. 4).

While the use of an API might reduce the performance of an application, a careful design of the API minimizes such an impact. Nevertheless, the benefits provided by our framework highly supersede a potential performance reduction, specially in off-line rendering environments. Once tested on different rendering systems and on a variety of scenes, one can decide to provide native implementations for specific rendering systems.

## 2 Related work

We begin by discussing meta-research systems in both graphics and vision which, like our own framework, have been developed to facilitate/improve the research process. Afterward, we focus on previous work on Monte Carlo denoising, which is the application that we use in our case study to illustrate the benefits of our framework.

### 2.1 Meta-research in graphics

Several systems have been proposed over the years to facilitate research development in graphics. Some of the most popular ones include Cg [22], Brook [8], and Halide [27]. Cg is a general-purpose, hardware-oriented, programming language and supporting system designed for the development of efficient GPU applications, and providing easy integration with the two major 3D graphics APIs (OpenGL and Direct 3D). Brook [8] is also a system for general-purpose computation that allows developers to use programmable GPUs as streaming co-processors, while abstracting GPU architectural details. Halide [27] tries to optimize image-processing algorithms by decoupling the algorithm's description from its schedule. This allows for an algorithm to be described once, while specific schedules are provided for different target platforms (e.g., CPUs, GPUs, mobile devices). Automatic generation of optimized schedules in Halide has been addressed in a follow-up work [24].

While the primary goal of these systems is to generate efficient code while abstracting hardware details from developers, our focus is on decoupling Monte Carlo algorithms from rendering systems. This greatly simplifies the task of porting algorithms to multiple rendering systems, freeing developers from the burden of knowing implementation details of specific renderers to be able to perform integration. Our system also makes a wider range of scenes available for testing, providing a comprehensive, multi-rendering system benchmark for Monte Carlo algorithms. Recently, Anderson et al. [1] proposed an approach to compile sampling BRDFs for MC applications. Their method is complementary to our work, and one could potentially integrate both systems in the future to provide even more flexibility for rendering researchers.

### 2.2 Benchmarking systems in computer vision

Quantitative benchmarks have been proposed for several computer visions areas, including optical flow [3,5], dense two-frame stereo correspondence [36], and alpha matting [30]. These initiatives have provided independent tools for assessing the quality of the results produced by existing and new algorithms and have led to significant progress in these areas.

**Optical flow** Barron et al. [5] compared accuracy, reliability, and density of velocity measurements for several established optical-flow algorithms and showed that their performance could vary significantly from one technique to another. Baker et al. [3] proposed another benchmark for optical-flow algorithms that considers aspects not covered by Barron et al. These include sequences containing non-rigid motion, realistic synthetic images, high frame-rate video to study interpolation errors, and modified stereo sequences of static scenes. The authors have made their datasets and evaluation results publicly available and provide the option for one to submit his own results for evaluation [2]

**Stereo correspondence** The Middlebury benchmark [36] provided a taxonomy and evaluation for dense two-frame stereo correspondence algorithms. The datasets and evaluation are publicly available on the web, and anyone can submit results for evaluation [37].

**Alpha matting** Rhemann et al. [30] introduced a benchmark system for alpha matting techniques. The authors provide some training data and use a test dataset for which the corresponding ground truth has not been disclosed. Similarly to the optical-flow and dense stereo correspondence benchmarks mentioned before, the results are available on-line, and anyone can submit results for evaluation [29].

**Video matting** Erofeev et al. [13] extended the alpha matting benchmark to videos, supporting both objective and subjective evaluations of video matting techniques. Training and test datasets are provided, with results and submissions being available through the web [12].

Unlike such systems, ours goes beyond rating submitted results computed off-line. It provides an API that allows Monte Carlo algorithms to be tested with different rendering systems using a variety of scenes. Thus, it can compare different techniques across multiple rendering systems, something that was not previously possible without requiring the developer to create multiple implementations tailored to individual rendering systems.

### 2.3 Monte Carlo denoising algorithms

Although there has been a significant amount of work on reducing the variance of MC rendered images through sampling/reconstruction (see [26,42]), for brevity we shall only focus on previous post-processing approaches that filter *general* Monte Carlo noise (i.e., noise from any and all distributed effects, path-tracing, and so on).

Soon after the seminal paper by Cook et al. [10] raised the problem of MC noise, there was some early work in general MC filtering, including approaches using nonlinear median and alpha-trimmed mean filters for edge-aware spike removal [20] and variable-width filter kernels to preserve energy and salient details [34]. However, in the years that followed, researchers largely ignored general MC filtering algorithms in favor of other variance reduction techniques, due to the inability of these filters to successfully remove the MC noise while preserving scene detail.

Recently, interest in general MC filtering algorithms has enjoyed a significant revival when Sen and Darabi [38,39] demonstrated that general, post-process image-space filters could effectively distinguish between noisy scene detail and MC noise and perform high-quality MC denoising. To do this, they used mutual information to determine dependencies between random parameters and scene features, and combined these dependencies to weight a cross-bilateral filter at each pixel in the image. Rousselle et al. [32] proposed to use a non-local means filter to remove general MC noise. Kalantari and Sen [19] applied median absolute deviation to estimate the noise level at every pixel to use any image denoising technique for filtering the MC noise. Finally, Delbracio et al. [11] modified the non-local means filter to use the color histograms of patches, rather than the noisy color patches, in the distance function.

Other approaches have effectively used error estimation for filtering general distributed effects. For example, Rousselle et al. [31] used error estimates to select different filter scales for every pixel to minimize reconstruction error. Furthermore, Li et al. [21] proposed to use Stein's unbiased risk estimator (SURE) [40] to select the best parameter for the spatial term of a cross-bilateral filter. Rousselle et al. [33] extended this idea to apply the SURE metric to choose the best of three candidate filters. Moon et al. [23] estimated the error for discrete sets of filter parameters using a weighted local regression. Bauszat et al. [6] posed the filter selection problem as an optimization and solved it with graph cuts. Bitterli et al. [7] used feature prefiltering, NL-Means regression weights and a collaborative denoising scheme to create an improved first-order regression technique.

More recently, Kalantari et al. [18] introduced a machine learning approach in which a neural network is used to drive the MC filter. This work was later extended by Bako et al. [4] by training end-to-end to produce high-quality denoising for production environments, and by Chaitanya et al. [9] for removing global illumination noise in interactive rendering.

All of these techniques have strengths and weaknesses in terms of the scene features they can satisfactorily handle, memory costs, execution time, etc. All these variables make a direct comparison of the various algorithms difficult. Our framework is intended to fill this gap by enabling easy deployment of an algorithm across multiple rendering systems through our API. We hope our system will help developers better understand the interplay among the various involved elements and available metrics, shedding some light on the occasional situations in which publications seem to disagree about the quality rank of different techniques.

## 3 System design

A physically based rendering system has to perform several tasks in order to generate an image. These include, for instance, reading the scene description file, building the internal scene representation data structure, generating well-distributed sampling positions in a high-dimensional space, computing global illumination, evaluating shading, reconstructing the final image, and saving the result. Several of those tasks hide a significant amount of complexity.

A good renderer implementation usually employs design practices that allow some level of extensibility. For example, it is common practice to facilitate adding new materials, shapes, cameras, samplers, reconstruction filters, and integrators, given they obey predefined interfaces. However, although different renderers use similar abstractions, implementing a new technique (e.g., a new denoising filter) still requires choosing a particular renderer (e.g., PBRT-v3 or Mitsuba). *This makes it hard to compare techniques implemented in different systems, and time-consuming to implement a technique in multiple rendering systems*.

Our framework seeks to avoid these limitations by decoupling the implementation of a new technique from any specific rendering system. For this, it hides sample value

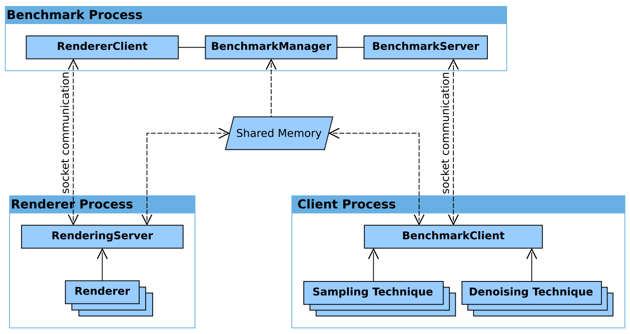


**Fig. 1** Main components of the system

computation details associated to individual rendering systems behind a general sampling evaluation interface. Thus, it allows for any technique to be seamlessly integrated with different rendering systems and provides a direct and simple mechanism for comparing different techniques' results on multiple (rendering) systems.

MC denoising techniques [18,33,39] are responsible for reconstructing images from sample values computed by the renderer while removing any residual noise from the MC estimation process. As such, they are a key component of all Monte Carlo rendering systems. Thus, we have chosen MC denoising algorithms to demonstrate the effectiveness of our framework. One should note, however, that our system can be used to implement/evaluate other MC techniques, such as general sampling and reconstruction algorithms. In our case study, we consider both adaptive and non-adaptive MC denoising techniques. Next, we describe the main components of our system and discuss how they are used to support the integration of techniques and rendering systems, and to perform benchmarks.

## 3.1 Main components

The architecture of the proposed system has three main components (Fig. 1): the *client process*, the *benchmark process*, and the *renderer process*. The client process implements the *technique* being integrated. The renderer process interfaces with the actual rendering system to evaluate the samples requested by the client process. The benchmark process controls the overall execution and saves the final image along with useful metadata information, such as reconstruction and rendering times.

Figure 2 shows a sequence diagram of a typical execution of the system. The benchmark process is executed, receiving a list of scenes to be rendered, each scene with a list of different sample budgets. For a given scene, the renderer process is executed. The benchmark process forwards the client's request to the renderer process and keeps track of the execution time and sample budget limits. Once the client process receives the requested samples, it reconstructs the

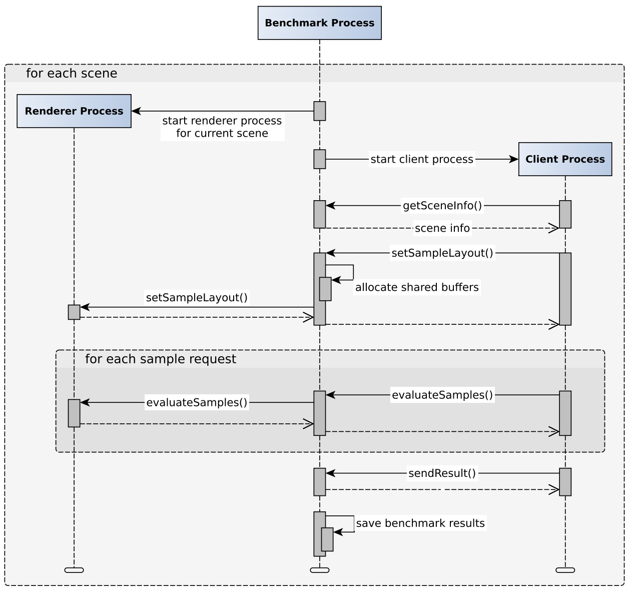


**Fig. 2** A typical execution of the system

final image and sends it to the benchmark process. The cycle can start again with another request (e.g., for a different sample density for the same scene, a different scene, a different technique, etc.). Note that we use the expression "benchmark process" to refer to this intermediate layer regardless of the system being used to locally evaluate a single or multiple techniques, or to perform or complement an actual full benchmark. The system also provides a web-based graphical user interface (GUI) for visual exploration of the results.

The separation between client and renderer processes allows us to provide a clean API to the client process, simplifying the task of implementing a new technique. Once a new technique is implemented using this renderer-agnostic API, it can be readily tested on a variety of scenes and compared against other techniques.

On the renderer side, this separation allows us to provide different renderers as back-ends to the system. When rendering a scene, the client process does not need to know the specific renderer being used. This also test the robustness of the technique to variations in sample values computed by different renderers.

### 3.1.1 Client process

The system expects techniques to follow the template shown in Fig. 3. Such a flow is general enough to cover a large variety of techniques, including MC denoising—adaptive, non-adaptive, a priori and a posteriori [42]—as well as sampling techniques. If a particular technique does not provide sampling positions, the renderer transparently supplies them.

When the client process starts, it is given a sample budget. In the *initial sampling* step (Fig. 3), the technique decides

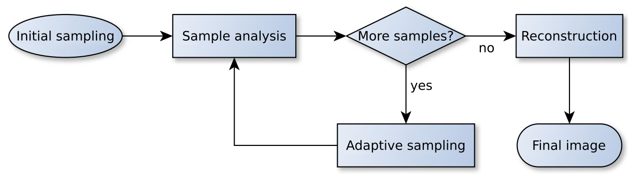


**Fig. 3** Template for techniques supported by our system

what portion of the sample budget to spend initially. If the technique is non-adaptive, the entire budget is spent in this step. Otherwise, one or more iterations of *sample analysis* and *adaptive sampling* are performed, until the sample budget is completely consumed. After the final image is reconstructed, the client process finishes. Besides the sample budget, the client has access to more information about the scene through a scene information querying API. This information allows the technique to adjust its parameters depending on the characteristics of the scene.

Our framework is general enough to support advanced techniques with adaptive sampling, allowing them to generate sample positions based on information from previous iterations. If the technique does not perform adaptive sampling, the renderer itself generates the sample positions. The API also allows the technique to specify which features it needs for each sample. Some may require only color information, while others may require geometric features like normals and depth. The technique also specifies the exact layout of sample components in memory.

#### 3.1.2 Benchmark process

The benchmark process manages the system execution and mediates the communication between client and renderer processes (Fig. 2). It is responsible for starting the renderer process, providing information about the sample layout and additional rendering parameters, and later collecting the computed samples to be forwarded to the client technique. The benchmark process also keeps track of the current sample budget, client process execution time, and saves the image reconstructed by the client process along with an execution log.

#### 3.1.3 Renderer process

It consists of a common rendering system that has been instrumented to communicate with the benchmark process. It is responsible for computing the samples needed by the client process, as well as providing information about the current loaded scene. To help instrumenting existing rendering systems, we provide a few auxiliary classes that implement the necessary API and help collecting the sample data throughout the system.

### 3.2 Scenes

Our system includes two general categories of scenes: *production*, and *experimental*. The first category includes scenes one would usually find in a production environment (e.g., most scenes shown in Fig. 10). They usually contain more detailed geometry and textures, a bigger variety of illumination settings, and aesthetically pleasing results. The second category includes scenes designed specifically to stress certain aspects of the filters. Figure 6 shows examples of experimental scenes. By including a variety of scenes in both categories, we hope to avoid biases when comparing different techniques.

When evaluating a scene, we consider two main aspects: *features* and *noise sources*. Features are legitimate details that denoising techniques must preserve, like textures and materials, geometric details, shading highlights. Noise sources are elements that introduce undesired noise artifacts, like camera effects (motion blur and depth-of-field), glossy materials, area lights, and indirect illumination.

### 3.3 Implementation details

Instrumenting additional rendering systems for use with our framework only requires implementing the endpoints needed to communicate with the benchmark process. We provide an auxiliary class called `RenderingServer`, which implements the communication protocol and exposes a higher-level API using a signal-slot mechanism. The `RenderingServer` class and a few other auxiliary classes make it easy to instrument a renderer.

Synchronization, control messages, and small messages are implemented using TCP socket messages on a predefined local port. Large buffers use shared memory, saving memory and avoiding transfer overhead. They are used to transfer samples to the client, and the reconstructed image to the benchmark process (Fig. 1).

## 4 Results

We have implemented our framework in C++. As a proof of concept, we have ported three well-known renderers, PBRT-v2 [25], PBRT-v3 [26], and Mitsuba [16], plus a procedural renderer to work as back-ends of our system. We have also adapted many state-of-the-art MC denoising methods to run on our framework: LWR [28], NFOR [7], LBF [18], RPF [39], SBF [21], RHF [11], NLM [32], RDFC [33], and GEM [31]. For this, we have instrumented the original source code provided by the rendering systems' developers and by the authors of these techniques with calls to our API. In the case of NFOR, we could not get the source code and implemented it from scratch. All results shown in the paper were gener-

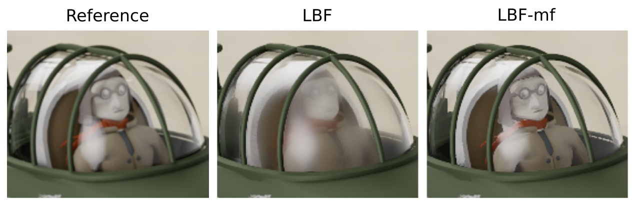


**Fig. 4** Rendering using geometric features. Reference image (left). Overblurring on transmitted scene details caused by relying on features at the first intersection point (center). Using features from the first non-specular intersection allows the denoiser to preserve those details (right)

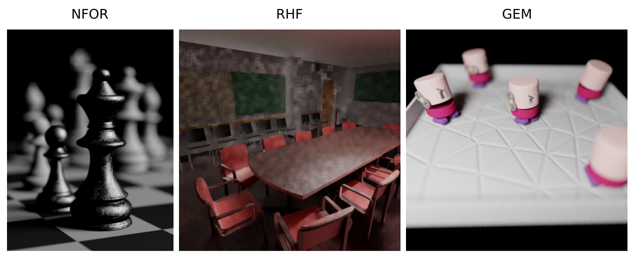


**Fig. 5** Images generated with our system using PBRT-v2 and the techniques NFOR, RHF, and GEM, respectively

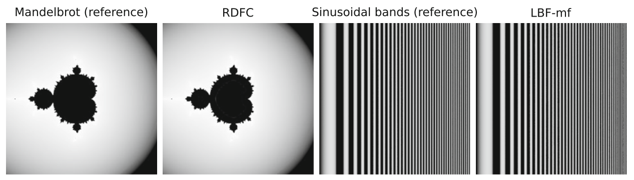


**Fig. 6** Examples of experimental scenes rendered with our system using a procedural renderer. Mandelbrot set (left). Increasing sinusoidal bands ($\sin(x^2)$) (right)

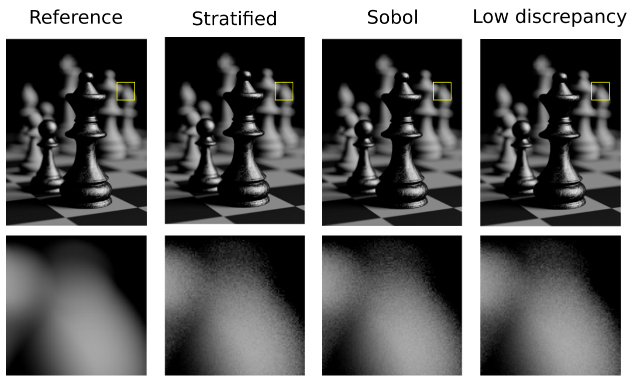


**Fig. 7** Examples of images produced by different sampling techniques using our system with the PBRT-v2 (top). The reference image was generated with 811,008 spp, while the stratified, Sobol, and low-discrepancy images used 64 spp. For comparison, the bottom row shows zoomed-in versions of the highlighted regions shown on top

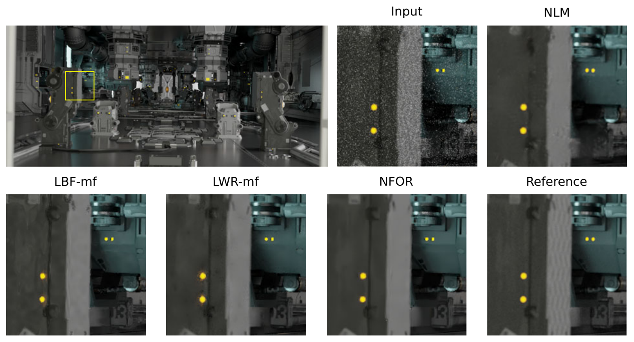


**Fig. 8** Texture details in the specular component of some materials (see the checker pattern on the light gray rectangle in the reference image) are not part of the "albedo" feature, making the denoisers to remove such details

ated on a 4 GHz i7-4790K CPU with 32 GB of RAM. This section provides several examples illustrating the use of the four rendering systems and eight state-of-the-art MC denoising algorithms. We also demonstrate the support for sampling techniques by adapting three commonly used ones: stratified, Sobol, and low discrepancy.

Some techniques use geometric features from the first intersection point to help them preserve scene details. This strategy tends to perform poorly on scenes with transparent glass and mirrors, as shown in Fig. 4 (center). To make comparisons among techniques fairer, we implemented modified versions of these techniques using the first non-specular intersection point instead. We indicate the modified versions by a suffix "-mf" (modified features)—Fig. 4 (right).

Figure 10 shows results of a benchmark created with seven MC denoising techniques and nine scenes from our scene pool. The scenes were selected as to form a representative set of situations that can challenge a denoiser. *Measure One* contains several glossy highlights, and *Measure One Moving* adds motion blur on top of that. *Crown in Glass* contains intricate bumpy textures with sources of caustics, all behind a layer of glass. *Furry Bunny* and *Curly Hair* contain fine geometric features that can easily be overblurred. *Bathroom* is a typical interior scene with several fine textures reflected by mirrors. *Country Kitchen Night* is a challenging global illumination scene with hidden light sources, being prone to fireflies (artifacts consisting of bright single pixels scattered over the image). Finally, *Glass of Water* is a mostly specular scene with many specular highlights.

The first row of Fig. 10 shows thumbnails of the ground truth images for the selected scenes. Although the image resolutions vary, their typical size is about 1024 × 1024 pixels. A small square highlights a challenging region in each scene. The corresponding regions for the noisy result, for the outputs generated by the various denoising techniques, and for the reference images are shown in the subsequent rows. From the scenes shown in Fig. 10, *Bathroom*, *Glass of Water*, and *Country Kitchen Night* were rendered using Mitsuba; the remaining six were rendered using PBRT-v3. Figures 5 and

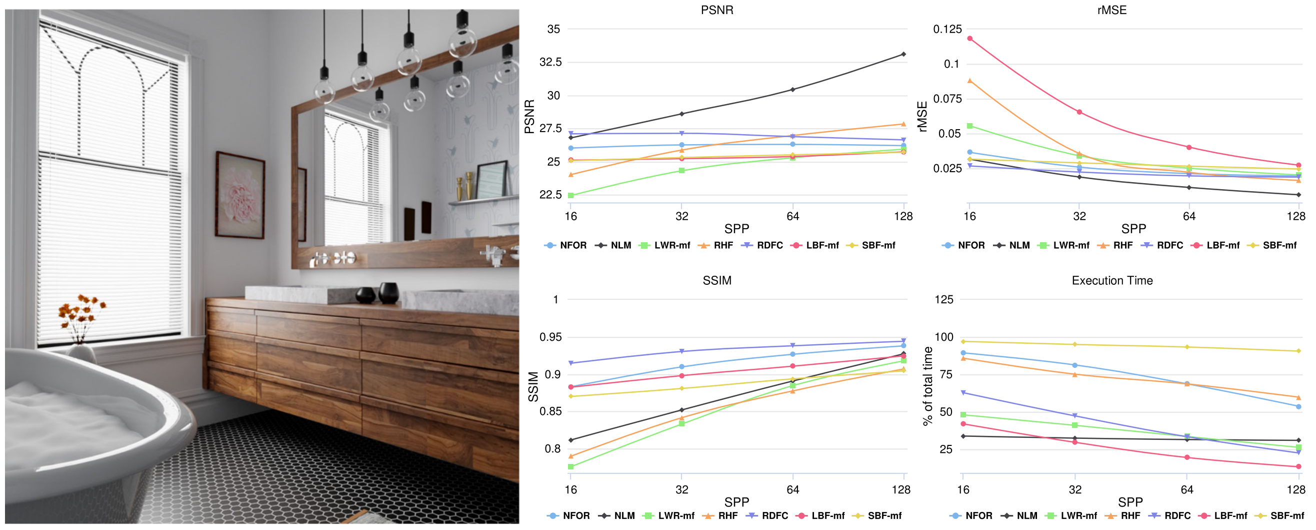


**Fig. 9** Use of our system's GUI for interactive exploration of the quantitative results generated by a benchmark

6 show examples of images generated with our framework using PBRT-v2 and a procedural renderer, respectively.

Our system can be used with and provides support for testing and comparing different sampling strategies. Figure 7 shows a scene rendered using our framework with three sampling techniques: stratified, Sobol, and low discrepancy. The images were generated using PBRT-v2, with 64 SPP. The reference image was rendered using 811,008 spp.

The results in Fig. 10 show that all techniques have some degree of trouble with glossy highlights, as shown in the scene *Measure One*. The glossy highlights are often overblurred or contain patchy artifacts. Glossy highlights are troublesome because the extra features used by the denoisers to tell legitimate scene details from noise do not help detecting the highlights. Another instance of this problem can be seen in Fig. 8. The subtle checker patterns seen on the reference image [Fig. 8 (bottom right)] come from a texture applied to the specular component of the material. This specular component is not part of the albedo feature used by denoisers, causing them the remove the detail.

Back to Fig. 10, scene *Measure One Moving* is a motion blur version of the previous scene. The strong motion blur effect makes the overblurring of the glossy highlights less visible, but it may also lend to other situations that may cause denoisers to produce overblur. All techniques have trouble preserving the fine motion blur details over the noisy glossy background.

The *Crown in Glass* scene contains bump-mapping details behind a layer of glass. Some techniques do a good job at preserving these details on the less noisy areas (e.g., NLM and RDFC). In darker, noisier regions, all techniques introduce some degree of overblurring.

Very fine geometry details, as commonly found in hair (*Curly Hair*) and fur (*Furry Bunny*) is also a frequent source of problems. Notice that even denoisers that rely on geometric features, as in the case with LBF, can overblur these details—although hair and fur are being captured by the geometric features, the sub-pixel-level detail in the presence of noise constitutes a challenge.

Scenes with very challenging illumination conditions—which translates to high levels of noise—are also problematic. High-energy spikes (fireflies) are very difficult to spread out while preserving energy, causing blob artifacts. As the *Country Kitchen Night* scene example shows, some techniques like *RDFC* do a good job at spreading fireflies, but small variations in the geometry of the scene can cause artifacts.

The results shown in Fig. 10 and the previous discussion illustrate the potential of our framework to provide qualitative assessments of MC denoising techniques, as well as to identify potential limitations of current approaches. As such, our system provides relevant information for guiding future research in the area. Our framework also contains a GUI for interactive exploration of benchmark results, which include quantitative assessments based on several metrics (MSE, rMSE, PSNR, SSIM, and execution time—Fig. 9). Table 1 shows the values of rMSE, PSNR, and SSIM for all examples in Fig. 10.

The proposed system is available for download in our project website.[1] The interactive version of the benchmark results corresponding to Fig. 10, using our web-based GUI, is also available. We would like to encourage the reader to explore such material. A video providing a brief tutorial on how to interactively explore the benchmark results can be found in the project website. Our GUI can be used with the results obtained for any technique that uses our framework.

[1] https://doi.org/10.7919/F46H4FGW.

**Table 1** Quantitative results for the images shown in Fig. 10 according to the rMSE, PSNR, and SSIM metrics

| SSP | NLM | | | LBF-mf | | | RHF | | | NFOR | | | LWR-mf | | | RDFC | | | SBF-mf | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | rMSE | PSNR | SSIM | rMSE | PSNR | SSIM | rMSE | PSNR | SSIM | rMSE | PSNR | SSIM | rMSE | PSNR | SSIM | rMSE | PSNR | SSIM | rMSE | PSNR | SSIM |
| Curly Hair | | | | | | | | | | | | | | | | | | | | | |
| 16 | 0.0068 | 33.4182 | 0.8559 | 0.0088 | 32.3764 | 0.8240 | 0.0097 | 32.0924 | 0.8249 | 0.0055 | 34.0426 | 0.8894 | 0.0096 | 32.1409 | 0.8138 | 0.0076 | 32.8546 | 0.8428 | 0.0094 | 32.0692 | 0.8239 |
| 32 | 0.0048 | 34.8156 | 0.8879 | 0.0067 | 33.3657 | 0.8527 | 0.0071 | 33.3176 | 0.8558 | 0.0041 | 35.2452 | 0.9114 | 0.0083 | 32.7106 | 0.8265 | 0.0058 | 33.9093 | 0.8700 | 0.0072 | 33.1470 | 0.8457 |
| 64 | 0.0032 | 36.6072 | 0.9200 | 0.0051 | 34.3698 | 0.8843 | 0.0050 | 34.8007 | 0.8892 | 0.0031 | 36.4066 | 0.9294 | 0.0066 | 33.6052 | 0.8502 | 0.0043 | 35.1458 | 0.8985 | 0.0057 | 34.1826 | 0.8676 |
| 128 | 0.0021 | 38.4619 | 0.9451 | 0.0043 | 35.0522 | 0.9076 | 0.0032 | 36.5669 | 0.9207 | 0.0024 | 37.5243 | 0.9442 | 0.0048 | 34.8144 | 0.8817 | 0.0031 | 36.5432 | 0.9248 | 0.0045 | 35.1782 | 0.8877 |
| Measure One | | | | | | | | | | | | | | | | | | | | | |
| 16 | 0.0114 | 29.9136 | 0.8230 | 0.0145 | 29.0600 | 0.8061 | 0.0111 | 30.1186 | 0.8066 | 0.0095 | 30.2342 | 0.8614 | 0.0150 | 28.3846 | 0.8136 | 0.0104 | 30.5375 | 0.8602 | 0.0113 | 29.8422 | 0.8251 |
| 32 | 0.0087 | 31.0691 | 0.8501 | 0.0112 | 30.0560 | 0.8267 | 0.0076 | 31.7795 | 0.8569 | 0.0068 | 31.6495 | 0.8882 | 0.0097 | 30.1050 | 0.8497 | 0.0071 | 31.9288 | 0.8865 | 0.0085 | 30.9448 | 0.8530 |
| 64 | 0.0065 | 32.3305 | 0.8737 | 0.0082 | 31.3141 | 0.8555 | 0.0054 | 33.2394 | 0.8899 | 0.0050 | 33.0332 | 0.9089 | 0.0064 | 31.8815 | 0.8806 | 0.0052 | 33.2241 | 0.9070 | 0.0063 | 32.0505 | 0.8744 |
| 128 | 0.0046 | 33.7154 | 0.8971 | 0.0057 | 32.7334 | 0.8851 | 0.0038 | 34.7063 | 0.9138 | 0.0037 | 34.3675 | 0.9246 | 0.0044 | 33.5219 | 0.9053 | 0.0037 | 34.5333 | 0.9239 | 0.0047 | 33.1541 | 0.8814 |
| Measure One Moving | | | | | | | | | | | | | | | | | | | | | |
| 16 | 0.0065 | 32.6101 | 0.8931 | 0.0117 | 30.5025 | 0.8228 | 0.0077 | 31.8505 | 0.8472 | 0.0073 | 31.9393 | 0.8967 | 0.0103 | 30.4283 | 0.8831 | 0.0061 | 32.8040 | 0.9096 | 0.0082 | 31.4845 | 0.8365 |
| 32 | 0.0048 | 33.8975 | 0.9093 | 0.0082 | 31.9737 | 0.8444 | 0.0049 | 33.9239 | 0.8940 | 0.0046 | 33.8510 | 0.9220 | 0.0069 | 32.2812 | 0.9021 | 0.0041 | 34.4503 | 0.9275 | 0.0057 | 33.0608 | 0.8808 |
| 64 | 0.0034 | 35.3087 | 0.9226 | 0.0057 | 33.3805 | 0.8671 | 0.0032 | 35.8296 | 0.9225 | 0.0031 | 35.5887 | 0.9382 | 0.0049 | 33.9451 | 0.9180 | 0.0029 | 36.0028 | 0.9404 | 0.0041 | 34.5003 | 0.9069 |
| 128 | 0.0024 | 36.9161 | 0.9363 | 0.0039 | 34.9353 | 0.8939 | 0.0021 | 37.6765 | 0.9406 | 0.0020 | 37.5113 | 0.9509 | 0.0034 | 35.6560 | 0.9319 | 0.0020 | 37.7502 | 0.9511 | 0.0029 | 36.0110 | 0.9122 |
| Bathroom | | | | | | | | | | | | | | | | | | | | | |
| 16 | 0.0318 | 26.8069 | 0.8119 | 0.1184 | 25.1307 | 0.8826 | 0.0882 | 24.0379 | 0.7900 | 0.0367 | 26.0398 | 0.8831 | 0.0557 | 22.4632 | 0.7759 | 0.0269 | 27.1138 | 0.9148 | 0.0318 | 25.0894 | 0.8702 |
| 32 | 0.0189 | 28.6246 | 0.8522 | 0.0656 | 25.2398 | 0.8981 | 0.0358 | 25.8946 | 0.8418 | 0.0261 | 26.2783 | 0.9098 | 0.0342 | 24.3322 | 0.8333 | 0.0225 | 27.1502 | 0.9305 | 0.0291 | 25.3261 | 0.8809 |
| 64 | 0.0115 | 30.4638 | 0.8910 | 0.0404 | 25.4054 | 0.9108 | 0.0223 | 26.9899 | 0.8778 | 0.0214 | 26.3141 | 0.9267 | 0.0253 | 25.3100 | 0.8848 | 0.0198 | 26.9017 | 0.9381 | 0.0267 | 25.5323 | 0.8938 |
| 128 | 0.0062 | 33.1411 | 0.9275 | 0.0274 | 25.7599 | 0.9246 | 0.0165 | 27.8578 | 0.9071 | 0.0197 | 26.2213 | 0.9380 | 0.0205 | 25.9545 | 0.9180 | 0.0189 | 26.6566 | 0.9441 | 0.0247 | 25.7199 | 0.9045 |
| Crown in Glass | | | | | | | | | | | | | | | | | | | | | |
| 16 | 0.0388 | 25.4848 | 0.7975 | 0.0716 | 22.3721 | 0.7458 | 0.0879 | 23.2257 | 0.7316 | 0.0508 | 24.5644 | 0.7813 | 0.0514 | 24.5724 | 0.7831 | 0.0383 | 25.8028 | 0.8292 | 0.1046 | 23.0870 | 0.6812 |
| 32 | 0.0233 | 27.6083 | 0.8460 | 0.0496 | 23.7700 | 0.7866 | 0.0473 | 25.3741 | 0.7976 | 0.0393 | 25.6622 | 0.8131 | 0.0325 | 26.4063 | 0.8281 | 0.0278 | 27.1110 | 0.8571 | 0.0707 | 24.6677 | 0.7401 |
| 64 | 0.0152 | 29.3789 | 0.8776 | 0.0325 | 25.7598 | 0.8290 | 0.0246 | 27.7020 | 0.8483 | 0.0276 | 26.9539 | 0.8413 | 0.0217 | 27.9946 | 0.8586 | 0.0198 | 28.4026 | 0.8780 | 0.0473 | 25.9973 | 0.7846 |
| 128 | 0.0105 | 31.0564 | 0.9022 | 0.0223 | 27.7330 | 0.8632 | 0.0140 | 29.9011 | 0.8854 | 0.0178 | 28.6235 | 0.8705 | 0.0145 | 29.5715 | 0.8813 | 0.0138 | 29.8623 | 0.8959 | 0.0288 | 27.4464 | 0.8259 |
| Furry Bunny | | | | | | | | | | | | | | | | | | | | | |
| 16 | 0.0125 | 30.7409 | 0.8577 | 0.0298 | 27.9409 | 0.8093 | 0.0175 | 29.7712 | 0.8426 | 0.0079 | 32.2501 | 0.9131 | 0.0211 | 29.0407 | 0.8246 | 0.0177 | 29.7484 | 0.8461 | 0.0190 | 29.3658 | 0.8348 |
| 32 | 0.0082 | 32.3536 | 0.8948 | 0.0218 | 29.0784 | 0.8344 | 0.0116 | 31.3199 | 0.8791 | 0.0058 | 33.5862 | 0.9313 | 0.0185 | 29.6635 | 0.8395 | 0.0132 | 30.8930 | 0.8715 | 0.0125 | 30.9737 | 0.8718 |
| 64 | 0.0057 | 33.8900 | 0.9216 | 0.0134 | 30.8606 | 0.8704 | 0.0076 | 32.9388 | 0.9088 | 0.0043 | 34.8727 | 0.9457 | 0.0157 | 30.3354 | 0.8532 | 0.0090 | 32.3368 | 0.8991 | 0.0092 | 32.2647 | 0.8978 |
| 128 | 0.0040 | 35.3538 | 0.9422 | 0.0073 | 33.1554 | 0.9103 | 0.0050 | 34.5940 | 0.9328 | 0.0031 | 36.2011 | 0.9579 | 0.0126 | 31.1616 | 0.8685 | 0.0059 | 34.0126 | 0.9256 | 0.0067 | 33.4865 | 0.9173 |

**Table 1** continued

| SSP | NLM | | | LBF-mf | | | RHF | | | NFOR | | | LWR-mf | | | RDFC | | | SBF-mf | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | rMSE | PSNR | SSIM | rMSE | PSNR | SSIM | rMSE | PSNR | SSIM | rMSE | PSNR | SSIM | rMSE | PSNR | SSIM | rMSE | PSNR | SSIM | rMSE | PSNR | SSIM |
| Smoke | | | | | | | | | | | | | | | | | | | | | |
| 16 | 0.0005 | 44.1372 | 0.9807 | 0.0009 | 42.0877 | 0.9791 | 0.0007 | 43.0729 | 0.9784 | 0.0005 | 44.0520 | 0.9795 | 0.0006 | 43.3547 | 0.9761 | 0.0006 | 43.4443 | 0.9800 | 0.0006 | 43.5586 | 0.9786 |
| 32 | 0.0004 | 44.9731 | 0.9818 | 0.0007 | 42.8350 | 0.9803 | 0.0006 | 43.9832 | 0.9801 | 0.0004 | 45.1488 | 0.9812 | 0.0005 | 44.2466 | 0.9782 | 0.0005 | 44.1988 | 0.9811 | 0.0004 | 45.0087 | 0.9809 |
| 64 | 0.0004 | 45.4864 | 0.9825 | 0.0005 | 43.9138 | 0.9812 | 0.0005 | 44.8552 | 0.9813 | 0.0004 | 45.9017 | 0.9822 | 0.0004 | 45.0737 | 0.9799 | 0.0004 | 45.0065 | 0.9819 | 0.0004 | 45.8805 | 0.9822 |
| 128 | 0.0004 | 45.8068 | 0.9829 | 0.0005 | 44.4741 | 0.9817 | 0.0004 | 45.4951 | 0.9821 | 0.0003 | 46.3656 | 0.9827 | 0.0004 | 45.6013 | 0.9808 | 0.0004 | 45.6926 | 0.9825 | 0.0003 | 46.3678 | 0.9828 |
| Country Kitchen Night | | | | | | | | | | | | | | | | | | | | | |
| 16 | 0.0138 | 27.8739 | 0.8140 | 0.0133 | 28.5459 | 0.8743 | 0.0115 | 29.8702 | 0.8368 | 0.0146 | 28.1630 | 0.8335 | 0.0250 | 24.2337 | 0.7679 | 0.0112 | 30.4661 | 0.8835 | 0.0104 | 28.8515 | 0.8689 |
| 32 | 0.0098 | 29.2396 | 0.8340 | 0.0090 | 30.2983 | 0.8920 | 0.0068 | 31.8874 | 0.8766 | 0.0088 | 30.1918 | 0.8689 | 0.0185 | 25.3280 | 0.7845 | 0.0066 | 32.4528 | 0.9055 | 0.0083 | 29.3832 | 0.8827 |
| 64 | 0.0069 | 30.7094 | 0.8546 | 0.0060 | 31.8616 | 0.9046 | 0.0045 | 33.4613 | 0.9028 | 0.0056 | 31.9737 | 0.8964 | 0.0121 | 26.9291 | 0.8139 | 0.0039 | 34.1304 | 0.9220 | 0.0075 | 29.6946 | 0.8841 |
| 128 | 0.0047 | 32.4360 | 0.8775 | 0.0039 | 33.4580 | 0.9179 | 0.0029 | 35.1759 | 0.9226 | 0.0033 | 34.1158 | 0.9192 | 0.0076 | 28.8330 | 0.8464 | 0.0023 | 35.9318 | 0.9367 | 0.1017 | 30.4582 | 0.8927 |
| Glass of Water | | | | | | | | | | | | | | | | | | | | | |
| 16 | 0.0171 | 28.7763 | 0.9273 | 0.0755 | 25.0842 | 0.8977 | 0.0439 | 26.5674 | 0.9106 | 0.0435 | 26.1671 | 0.9067 | 0.0278 | 26.5721 | 0.9128 | 0.0349 | 27.0671 | 0.9206 | 0.0352 | 26.2145 | 0.8984 |
| 32 | 0.0115 | 30.5121 | 0.9457 | 0.0643 | 25.6950 | 0.9065 | 0.0324 | 27.3999 | 0.9271 | 0.0305 | 27.0536 | 0.9167 | 0.0209 | 27.8729 | 0.9265 | 0.0265 | 27.6904 | 0.9287 | 0.0272 | 27.2131 | 0.9124 |
| 64 | 0.0078 | 32.2051 | 0.9586 | 0.0587 | 25.9437 | 0.9128 | 0.0220 | 28.5058 | 0.9404 | 0.0247 | 27.6811 | 0.9266 | 0.0147 | 29.0346 | 0.9415 | 0.0215 | 28.4385 | 0.9381 | 0.0198 | 28.2481 | 0.9264 |
| 128 | 0.0052 | 33.8967 | 0.9691 | 0.0537 | 26.1498 | 0.9168 | 0.0156 | 29.7450 | 0.9536 | 0.0182 | 28.5106 | 0.9394 | 0.0105 | 30.0555 | 0.9537 | 0.0156 | 29.2368 | 0.9480 | 0.0180 | 28.6043 | 0.9313 |
| Averages | | | | | | | | | | | | | | | | | | | | | |
| 16 | 0.0155 | 31.0847 | 0.8623 | 0.0383 | 29.2334 | 0.8491 | 0.0309 | 30.0674 | 0.8410 | 0.0196 | 30.8281 | 0.8827 | 0.0241 | 29.0212 | 0.8390 | 0.0171 | 31.0932 | 0.8874 | 0.0256 | 29.9514 | 0.8464 |
| 32 | 0.0100 | 32.5659 | 0.8891 | 0.0263 | 30.2569 | 0.8691 | 0.0171 | 31.6533 | 0.8788 | 0.0140 | 32.0741 | 0.9047 | 0.0167 | 30.3274 | 0.8632 | 0.0127 | 32.1983 | 0.9065 | 0.0188 | 31.0806 | 0.8720 |
| 64 | 0.0067 | 34.0422 | 0.9114 | 0.0189 | 31.4233 | 0.8906 | 0.0106 | 33.1470 | 0.9068 | 0.0106 | 33.1917 | 0.9217 | 0.0120 | 31.5677 | 0.8867 | 0.0096 | 33.2877 | 0.9226 | 0.0141 | 32.0390 | 0.8909 |
| 128 | 0.0045 | 35.6427 | 0.9311 | 0.0143 | 32.6057 | 0.9112 | 0.0071 | 34.6354 | 0.9287 | 0.0078 | 34.3823 | 0.9364 | 0.0087 | 32.7966 | 0.9075 | 0.0073 | 34.4688 | 0.9370 | 0.0264 | 32.4659 | 0.9060 |

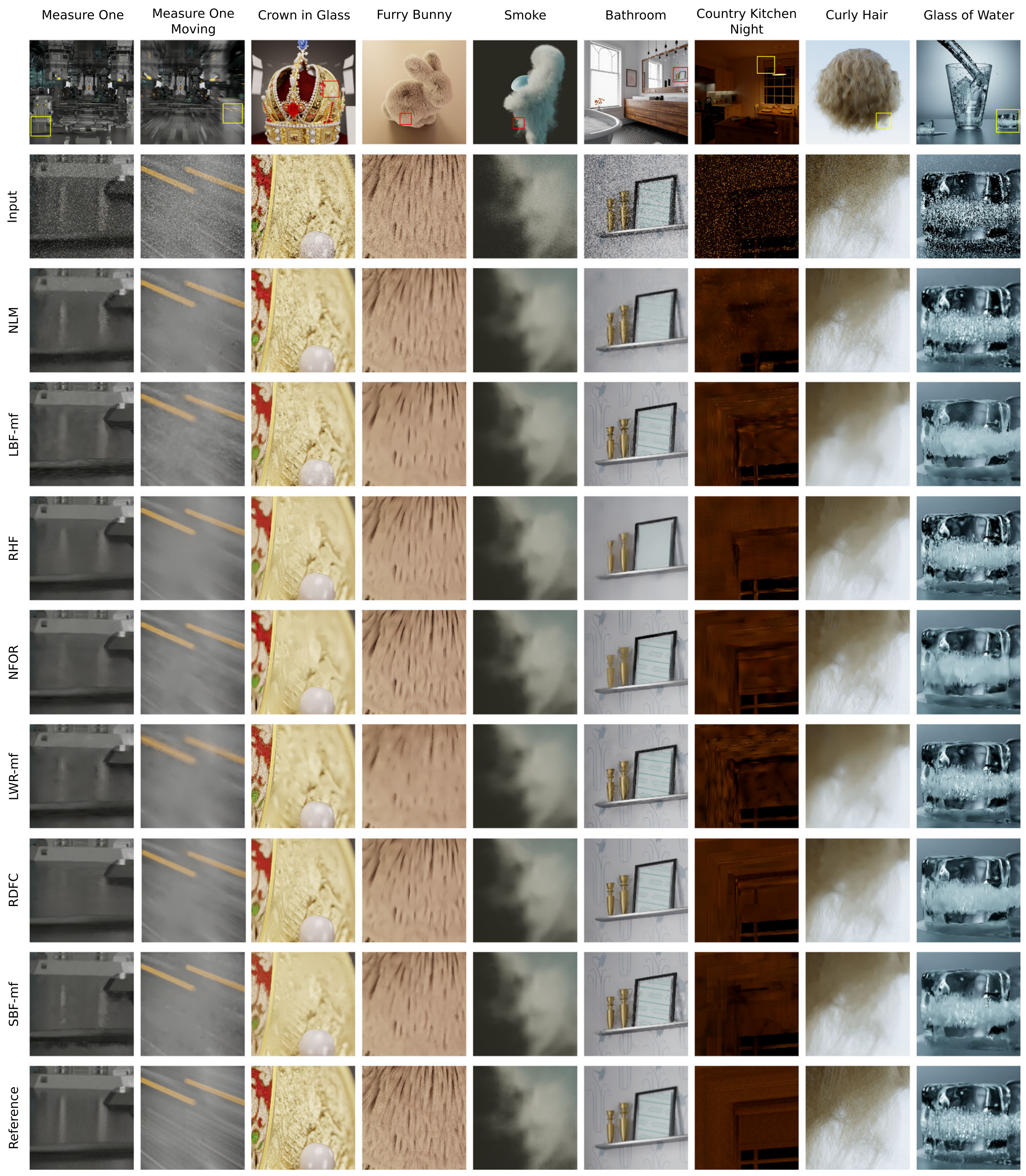


**Fig. 10** Results from a benchmark including seven MC denoising techniques and nine scenes (from our scene pool) that pose challenges to denoising methods. All results were generated with 128 samples per pixel

Figure 9 shows a snapshot of some of the quantitative information obtained when using our framework to render the *Bathroom* scene using the seven denoising techniques shown in Fig. 10. The graphs compare the performance of the techniques according to PSNR, rMSE, SSIM, and execution time for 16, 32, 64, and 128 spp.

## 4.1 Discussion

**Communication overhead** This can be significant depending on how a technique requests samples. If the entire budget is requested in a single call, the overhead is negligible, but if each call requests a single sample, as in the case of MDAS [14], the overhead becomes prohibitive for anything but a very small number of samples and image sizes.

**Memory overhead** The current implementation of our system requires that all samples be kept in memory at once, which imposes a limit on the maximum sample budget. We plan on avoiding this restriction by making blocks of samples available to the client process as soon as they are produced by the renderer.

**Scene file format** Rendering systems adopt proprietary scene file formats. In addition, certain features (e.g., materials) supported by one renderer might not be available for others. Thus, currently, in order to use a scene file with a different renderer, one has to convert it to the format used by the desired renderer.

## 5 Conclusion and future work

We have presented a novel framework for easy development and evaluation of sampling and denoising MC algorithms on multiple rendering systems. Conversely, it also allows for rendering systems to quickly incorporate new algorithm that conform to our API. This makes it straightforward to perform benchmarks involving various algorithms across different renderers. We have demonstrated the effectiveness of our system by instrumenting four rendering systems (PBRT-v3, PBRT-v2, Mitsuba, and a procedural renderer), eight state-of-the-art MC denoising algorithms, three sampling techniques, and by benchmarking these denoising algorithms on multiple renderers.

We have provided a qualitative assessment of the evaluated MC denoising techniques, identifying potential limitations of existing approaches. This information might guide future research in the area. The visual exploration of the quantitative data collected during the benchmark also provides valuable feedback for researchers and users, helping them to address the practical question of identifying the most effective techniques for rendering scenes with a given set of features.

### 5.1 Future work

We plan on releasing an on-line benchmarking service to allow researchers to submit their techniques for evaluation and ranking. This would be similar to other benchmark services, such as the Middlebury benchmark [36].

To solve the current memory overhead problem, we plan to group samples into smaller-sized chunks and send them to the client process as soon as they are computed by the renderer. This should increase the communication overhead but will make the memory complexity independent of the amount of requested samples, which is a fair tradeoff. We will make such an implementation available on the project website.

**Acknowledgements** The authors thank Matt Pharr, Greg Humphreys, and Wenzel Jakob for making the source code of PBRT-v2, PBRT-v3, and Mitsuba publicly available. We also thank the authors of the following techniques for kindly providing their source code: LBF, RHF, LWR, RDFC, RPF, SBF, NLM, GEM. The following individuals and institutions provided the scenes used in the paper: Martin Lubich (Crown), Cem Yuksel (Curly Hair), Jesper Lloyd (Toy Gyro), Wojciech Jarosz (Chess), Andrew Kensler (Toasters), Bernhard Vogl and Stanford CG Lab (Furry Bunny), Mareck (Bathroom), aXel (Glass of Water), Jay-Artist (Country Kitchen), Beeple (Measure One), Anat Grynberg and GregWard (Conference), Duc Nguyen, Ron Fedkiw, and Nolan Goodnight (Smoke).

**Funding** This work was funded by CAPES and CNPq-Brazil (fellowships and Grants 306196/2014-0 and 423673/2016-5), and US National Science Foundation Grants IIS-1321168 and IIS-1619376.

## Compliance with ethical standards

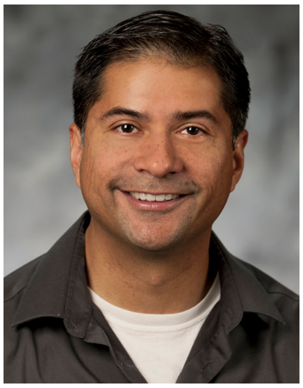
**Conflict of interest** The authors Jonas Deyson B. Santos, Pradeep Sen, and Manuel M. Oliveira declare they have no conflict of interest.

## References

1. Anderson, L., Li, T.-M., Lehtinen, J., Durand, F.: Aether: An embedded domain specific sampling language for Monte Carlo rendering. ACM Trans. Graph. **36**(4), 99:1–99:16 (2017). https://doi.org/10.1145/3072959.3073704
2. Baker, S., Scharstein, D., Lewis, J., Roth, S., Black, M., Szeliski, R.: Middlebury Flow Accuracy and Interpolation Evaluation. http://vision.middlebury.edu/flow/eval/ (2011). Accessed 4 Mar 2018
3. Baker, S., Scharstein, D., Lewis, J.P., Roth, S., Black, M.J., Szeliski, R.: A database and evaluation methodology for optical flow. IJCV **92**(1), 1–31 (2011)
4. Bako, S., Vogels, T., Mcwilliams, B., Meyer, M., Novák, J., Harvill, A., Sen, P., Derose, T., Rousselle, F.: Kernel-predicting convolutional networks for denoising Monte Carlo renderings. ACM Trans. Graph. **36**(4), 97:1–97:14 (2017)
5. Barron, J.L., Fleet, D.J., Beauchemin, S.S.: Performance of optical flow techniques. IJCV **12**(1), 43–77 (1994)
6. Bauszat, P., Eisemann, M., Eisemann, E., Magnor, M.: General and robust error estimation and reconstruction for Monte Carlo rendering. Comput. Graph. Forum **34**(2), 597–608 (2015)

7. Bitterli, B., Rousselle, F., Moon, B., Iglesias-Guitián, J.A., Adler, D., Mitchell, K., Jarosz, W., Novák, J.: Nonlinearly weighted first-order regression for denoising Monte Carlo renderings. Comput. Graph. Forum **35**(4), 107–117 (2016)
8. Buck, I., Foley, T., Horn, D., Sugerman, J., Fatahalian, K., Houston, M., Hanrahan, P.: Brook for GPUs: stream computing on graphics hardware. ACM Trans. Graph. **23**(3), 777–786 (2004)
9. Chaitanya, C.R.A., Kaplanyan, A.S., Schied, C., Salvi, M., Lefohn, A., Nowrouzezahrai, D., Aila, T.: Interactive reconstruction of Monte Carlo image sequences using a recurrent denoising autoencoder. ACM Trans. Graph. **36**(4), 98:1–98:12 (2017)
10. Cook, R.L., Porter, T., Carpenter, L.: Distributed ray tracing. In: Proceedings of SIGGRAPH '84, pp. 137–145
11. Delbracio, M., Musé, P., Buades, A., Chauvier, J., Phelps, N., Morel, J.M.: Boosting Monte Carlo rendering by ray histogram fusion. ACM Trans. Graph. **33**(1), 8:1–8:15 (2014)
12. Erofeev, M., Gitman, Y., Vatolin, D., Fedorov, A., Wang, J.: Videomatting. http://videomatting.com/ (2014). Accessed 4 Mar 2018
13. Erofeev, M., Gitman, Y., Vatolin, D., Fedorov, A., Wang, J.: Perceptually motivated benchmark for video matting. In: Xie, X., Jones, M.W., Tam, G.K.L. (eds.) BMVA Press Proceedings, pp. 99.1–99.12 (2015)
14. Hachisuka, T., Jarosz, W., Weistroffer, R.P., Dale, K., Humphreys, G., Zwicker, M., Jensen, H.W.: Multidimensional adaptive sampling and reconstruction for ray tracing. ACM TOG **27**(212), 1 (2008)
15. Heck, D., Schlömer, T., Deussen, O.: Blue noise sampling with controlled aliasing. ACM Trans. Graph. **32**(3), 25:1–25:12 (2013)
16. Jakob, W.: Mitsuba renderer. http://www.mitsuba-renderer.org (2010). Accessed 4 Mar 2018
17. Kajiya, J.T.: The rendering equation. SIGGRAPH'86 **20**(4), 143–150 (1986)
18. Kalantari, N.K., Bako, S., Sen, P.: A machine learning approach for filtering Monte Carlo noise. ACM Trans. Graph. **34**(4), 122:1–122:12 (2015)
19. Kalantari, N.K., Sen, P.: Removing the noise in Monte Carlo rendering with general image denoising algorithms. Comput. Graph. Forum **32**(2pt1), 93–102 (2013)
20. Lee, M.E., Redner, R.A.: Filtering: a note on the use of nonlinear filtering in computer graphics. IEEE Comput. Graph. Appl. **10**(3), 23–29 (1990)
21. Li, T.M., Wu, Yt, Chuang, Yy: SURE-based optimization for adaptive sampling and reconstruction. ACM Trans. Graph. **31**, 1 (2012)
22. Mark, W.R., Glanville, R.S., Akeley, K., Kilgard, M.J.: Cg: a system for programming graphics hardware in a c-like language. ACM Trans. Graph. **22**(3), 896–907 (2003)
23. Moon, B., Carr, N., Yoon, S.E.: Adaptive rendering based on weighted local regression. ACM Trans. Graph. **33**(5), 170:1–170:14 (2014)
24. Mullapudi, R.T., Adams, A., Sharlet, D., Ragan-Kelley, J., Fatahalian, K.: Automatically scheduling halide image processing pipelines. ACM Trans. Graph. **35**(4), 83:1–83:11 (2016)
25. Pharr, M., Humphreys, G.: Physically Based Rendering, from Theory to Implementation, 2nd edn. Morgan Kaufmann, Los Altos (2010)
26. Pharr, M., Jakob, W., Humphreys, G.: Physically Based Rendering, from Theory to Implementation, 3rd edn. Morgan Kaufmann, Los Altos (2016)
27. Ragan-Kelley, J., Adams, A., Paris, S., Levoy, M., Amarasinghe, S., Durand, F.: Decoupling algorithms from schedules for easy optimization of image processing pipelines. ACM Trans. Graph. **31**(4), 32:1–32:12 (2012)
28. Ren, P., Wang, J., Gong, M., Lin, S., Tong, X., Guo, B.: Global illumination with radiance regression functions. ACM Trans. Graph. **32**, 1 (2013)
29. Rhemann, C., Rother, C., Wang, J., Gelautz, M., Kohli, P., Rott, P.: Alpha matting evaluation website. http://www.alphamatting.com/eval_25.php (2009). Accessed 4 Mar 2018
30. Rhemann, C., Rother, C., Wang, J., Gelautz, M., Kohli, P., Rott, P.: A perceptually motivated online benchmark for image matting. In: CVPR, pp. 1826–1833 (2009)
31. Rousselle, F., Knaus, C., Zwicker, M.: Adaptive sampling and reconstruction using greedy error minimization. ACM Trans. Graph. **30**(6), 159:1–159:12 (2011)
32. Rousselle, F., Knaus, C., Zwicker, M.: Adaptive rendering with non-local means filtering. ACM Trans. Graph. **31**(6), 195:1–195:11 (2012)
33. Rousselle, F., Manzi, M., Zwicker, M.: Robust denoising using feature and color information. Comput. Graph. Forum **32**(7), 121–130 (2013)
34. Rushmeier, H.E., Ward, G.J.: Energy preserving non-linear filters. In: Proceedings of SIGGRAPH '94, pp. 131–138 (1994)
35. Samet, H.: Sorting in space: multidimensional, spatial, and metric data structures for computer graphics applications. In: ACM SIGGRAPH ASIA 2010 Courses, SA '10, pp. 3:1–3:52 (2010)
36. Scharstein, D., Szeliski, R.: A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. IJCV **47**(1–3), 7–42 (2002)
37. Scharstein, D., Szeliski, R., Hirschmüller, H.: Middlebury Stereo Vision Page. http://vision.middlebury.edu/stereo/ (2002). Accessed 4 Mar 2018
38. Sen, P., Darabi, S.: Implementation of Random Parameter Filtering. Tech. Rep. EECE-TR-11-0004, University of New Mexico (2011)
39. Sen, P., Darabi, S.: On filtering the noise from the random parameters in Monte Carlo rendering. ACM Trans. Graph. **31**(3), 1–15 (2012)
40. Stein, C.M.: Estimation of the mean of a multivariate normal distribution. Ann. Stat. **9**(6), 1135–1151 (1981)
41. Veach, E., Guibas, L.J.: Metropolis light transport. In: Proceedings of SIGGRAPH '97, pp. 65–76 (1997)
42. Zwicker, M., Jarosz, W., Lehtinen, J., Moon, B., Ramamoorthi, R., Rousselle, F., Sen, P., Soler, C., Yoon, S.E.: Recent advances in adaptive sampling and reconstruction for Monte Carlo rendering. Comput. Graph. Forum **34**(2), 667–681 (2015)

**Jonas Deyson Brito dos Santos** is a Ph.D. candidate at the Federal University of Rio Grande do Sul (UFRGS) in Brazil. He graduated in Computer Science from Federal University of Ceará (UFC) in 2010 and received his master's degree from the same university in 2013. His research interests consists in computer graphics, especially Monte Carlo rendering and denoising.



**Dr. Pradeep Sen** is an Associate Professor in the Department of Electrical and Computer Engineering at the University of California, Santa Barbara. He received his B.S. from Purdue University and his M.S. and Ph.D. from Stanford University. His core research is in the areas of computer graphics, computational image processing, computer vision, and machine learning. He received an NSF CAREER award, and was named a Sony Pictures Imageworks Faculty Fellow in 2008.

**Manuel M. Oliveira** is a faculty member at the Federal University of Rio Grande do Sul (UFRGS) in Brazil. He received his Ph.D. from the University of North Carolina at Chapel Hill in 2000. Before joining UFRGS in 2002, he was an assistant professor at SUNY Stony Brook (2000–2002). In the 2009–2010 academic year, he was a visiting associate professor at the MIT Media Lab. He received the Distinguished Researcher Award in Mathematics, Statistics, and Computer Science from the State of Rio Grande do Sul's Research Agency (FAPERGS) in 2016. His research interests cover most aspects of computer graphics, especially in the frontiers among graphics, image processing, and vision (both human and machine). He is an associate editor of IEEE CG&A and IEEE TVCG.