Image-driven local filter optimization for reconstruction of Monte Carlo images by novice users

Alexander Overvoorde

TU Delft

Image-driven local filter optimization for reconstruction of Monte Carlo images by novice users

by

Alexander Overvoorde

to obtain the degree of Master of Science

at the Delft University of Technology,

to be defended publicly on Wednesday November 8, 2017 at 2:00 PM.

Student number:	4153235	
Project duration:	February 1, 2017 – November 1, 2017	
Thesis committee:	Prof. dr. E. Eisemann,	TU Delft, supervisor
	Dr. P. Bauszat,	TU Delft, daily supervisor
	Dr. A. Vilanova,	TU Delft, department associate professor
	Dr. D. Tax,	TU Delft, external assistant professor

This thesis is confidential and cannot be made public until November 9, 2017.

An electronic version of this thesis is available at `http://repository.tudelft.nl/`.

TU Delft

Preface

Although I have had a natural fondness for anything related to computer science for as long as I can remember, computer graphics has always had a special place in my mind. To a large degree my passion for computer science is driven by a desire to understand what kind of algorithms bring the digital worlds in my favourite video games to life. Choosing to specialize in computer graphics for my master degree was a given, but it was quite tough to decide on a topic. Ultimately the courses taught by my supervisor Elmar Eisemann helped me steer towards Monte Carlo light transport techniques and combine it with an interesting image processing twist. Special thanks go out to my daily supervisor Pablo Bauszat for providing invaluable help in navigating this broad and complex topic. He was always available to explain algorithms, provide feedback on drafts and presentations, and to provide general support during the tough moments. Surprisingly, one of his most important contributions to my success was in fact a simple question: would you prefer to work at home or at a designated workspace at the university? This question lead me to the master students room at the Intelligent Systems department where I ended up spending my full nine months. This room provided by the university allowed computer science master students from a wide variety of departments to come together and work on their thesis while supporting each other with valuable advice and fun moments to unwind. The people in this small community provided me with great new insights from completely different perspectives and greatly enhanced my experience of working on my thesis. Lastly, I would like to thank my friends and family for their support and understanding during the most challenging moments. I could not have done it without the support of these amazing people.

Alexander Overvoorde
Delft, November 2017

Contents

1

Abstract

Path tracing has been successfully utilized in modern animated films to produce photorealistic images. However, with path tracing being a Monte Carlo method, a large amount of light paths must be sampled to reduce image noise to acceptable levels. Although a wide variety of acceleration and importance sampling techniques have been developed to reduce the cost of individual samples and reduce the number of required samples, path tracing remains a very computationally expensive approach. Monte Carlo error estimation and reconstruction approaches the problem from a signal processing perspective by reducing the sampling budget and instead applying image denoising filters to lower noise levels. Although these image denoising filters can be successfully applied to post-process images into near ground truth results, estimating the right parameters to use remains an open problem. We propose that limitations of automatic parameter estimation can be overcome by taking advantage of the ground truth image in the mind of artists. To that end we introduce a workflow and associated user interface to let artists locally tune and apply filters to noisy Monte Carlo images. The interface lets users explore the high dimensional parameter spaces of filters through an image navigation paradigm, which means that users pick the right filter and settings by choosing the end result they want. We organize a case study to evaluate the efficacy of interactive filter tuning and to compare the efficiency of slider-based parameter choice with the image navigation paradigm. With the results from the case study we show that our workflow allows even novice users with no prior image processing experience to quickly select appropriate filters. Users are able to produce higher quality results using our local parameter painting approach compared to global optimization in the same time span. Users also prefer the image navigation interface for filter choice and tuning to sliders, but we are only able to quantify the benefits on images that require filters with a high dimensional parameter space.

2

Introduction

Physically based rendering methods are increasing in popularity because they easily let artists achieve the results they want [11]. A prominent example of such rendering methods is Monte Carlo path tracing, which realistically simulates light to produce photorealistic images, as evident from Figure 2.1. Not only is its use in the digital animation industry continually increasing [29], but its qualities are further exemplified by IKEA's usage of path tracing to generate nearly 75% of the images in their furniture catalogs [44]. Video games have traditionally used rasterization for rendering, but are also gradually moving to hybrid approaches that incorporate ray tracing and path tracing for certain effects like global illumination [13]. Monte Carlo path tracing has a long history [45], but only recently has hardware caught up to produce computer generated images within reasonable time constraints.

2.1. Path Tracing

Path tracing is a Monte Carlo method for solving the rendering equation [26]. Similar to ray tracing, rays are cast from the camera into the scene through pixels in the image plane. When the ray intersects with a surface, the BRDF [42] of the surface material is used to generate a random reflection ray. This may always be a perfect reflection in case of a perfectly specular material, but may also be a random vector in the hemisphere oriented by the surface normal in case of a perfectly diffuse material. This reflection ray is intersected with the scene again and this process repeats to determine one possible path of how light could have bounced around the scene to arrive at the camera. The bouncing continues until the ray hits a light emitting surface or a specified threshold has been exceeded.

However, there are many possible light paths per pixel that can potentially produce a wide variance of colors, as demonstrated in Figure 2.2. A single one of these paths, a single *sample*, is not representative of all light interactions in the scene and will produce a very noise image. Akin to general statistics, the standard deviation of this noise can be reduced by evaluating the mean of multiple samples per pixel. However, the standard deviation decreases very slowly compared to the number of samples that are computed, as seen in Figure 2.3.

Upwards of thousands of samples may need to be calculated to reduce noise levels to such a degree that the quality of the images is acceptable for product catalogues and digital movies. Acceleration hardware like GPUs provide a high constant speedup, but they cannot eliminate the need for high amounts of samples.

2.2. Acceleration Techniques

Several avenues have been proposed for improving image quality given fixed time constraints. The computationally most expensive part of path tracing remains ray-scene intersection. The cost of these calculations can be reduced with acceleration structures [4], which can significantly improve performance in scenes with complex geometry. It is also possible to reduce the noise level by sampling more effectively. For example in a naive path tracer it is likely that many paths do not arrive at a light source within the bounce threshold. Instead of just following a path from the camera, in bi-directional path tracing [33], a path is also being

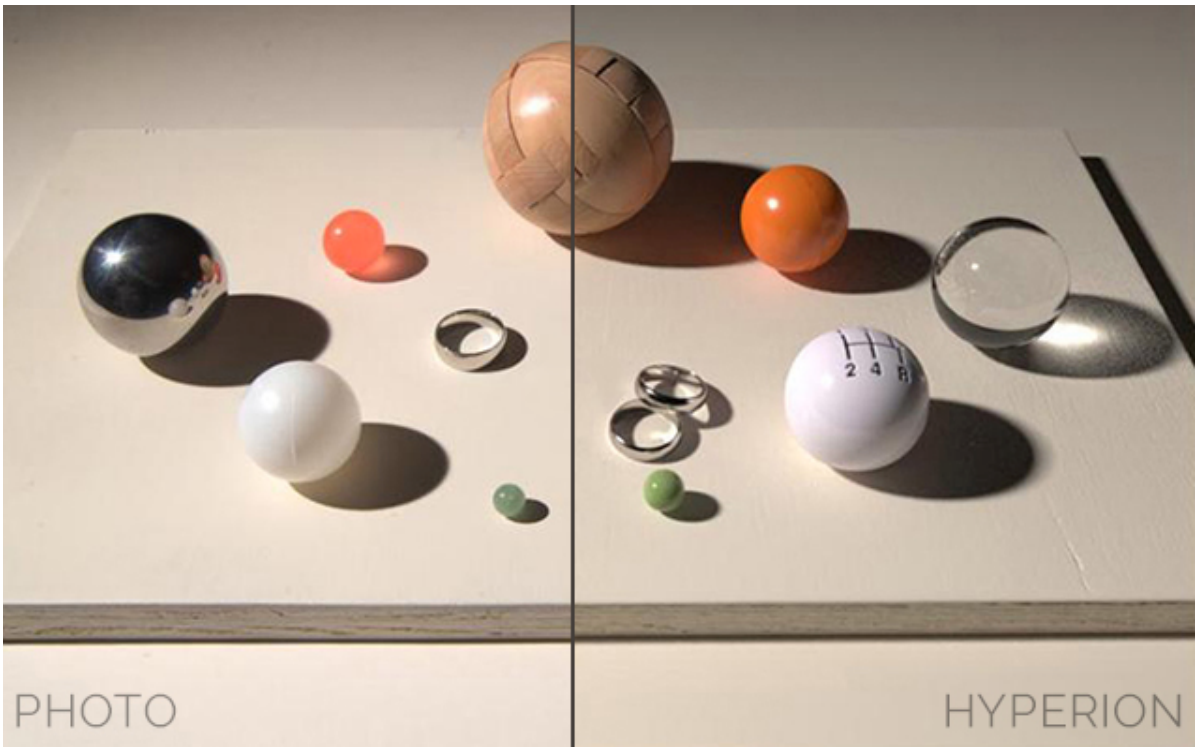


Figure 2.1: Comparison between a photograph and a path traced image from Disney's Hyperion renderer.

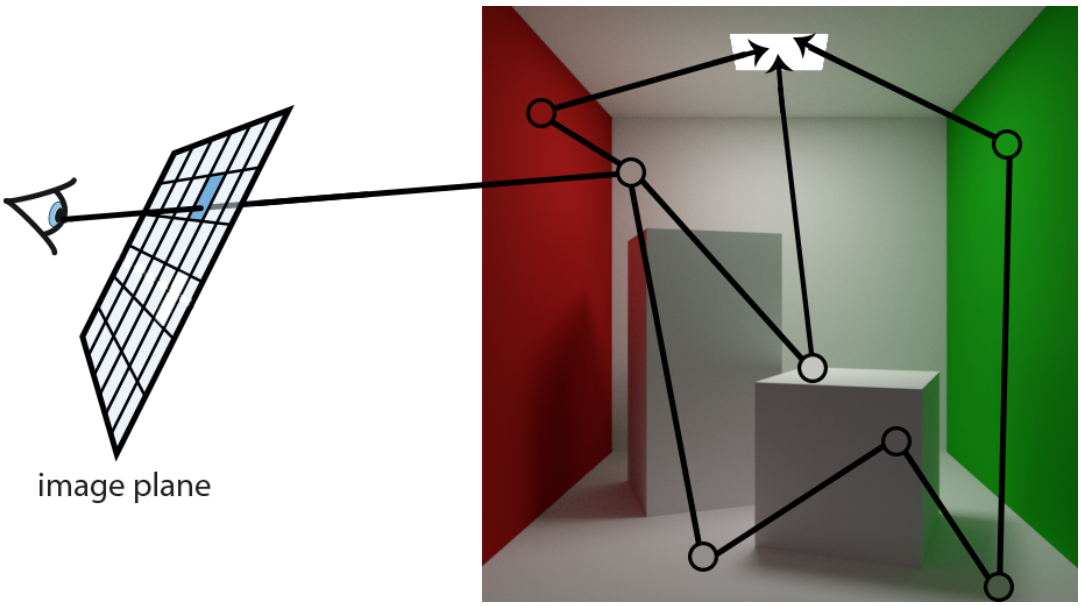


Figure 2.2: Examples of random paths that light could have been through the scene to arrive at a pixel in the image plane.

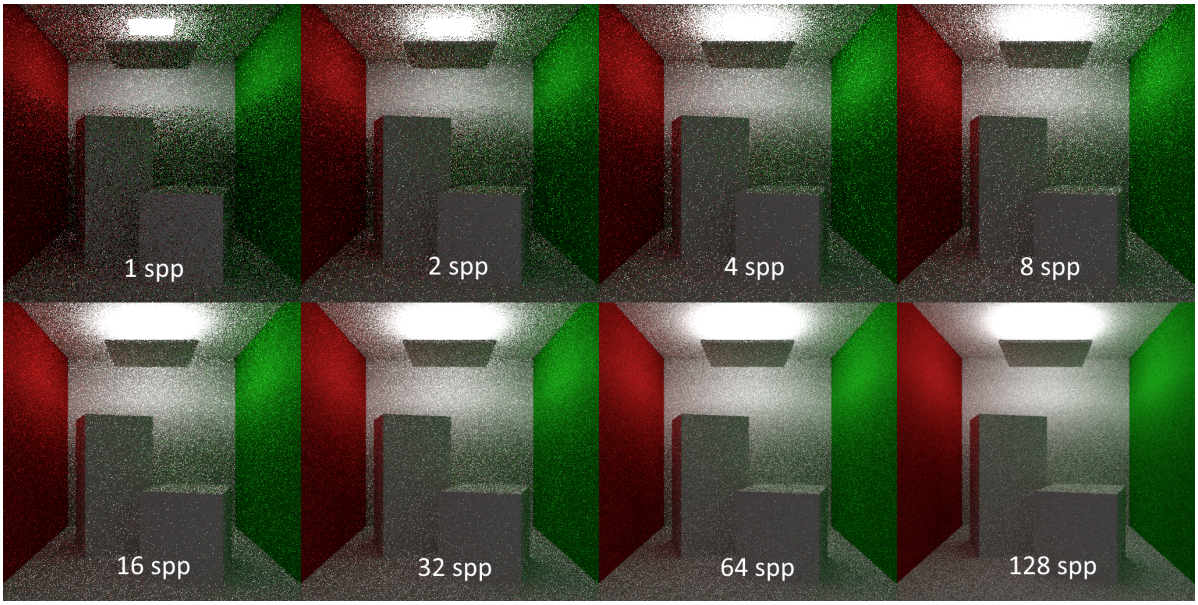


Figure 2.3: The standard deviation of Monte Carlo noise decreases slowly compared to the number of samples.

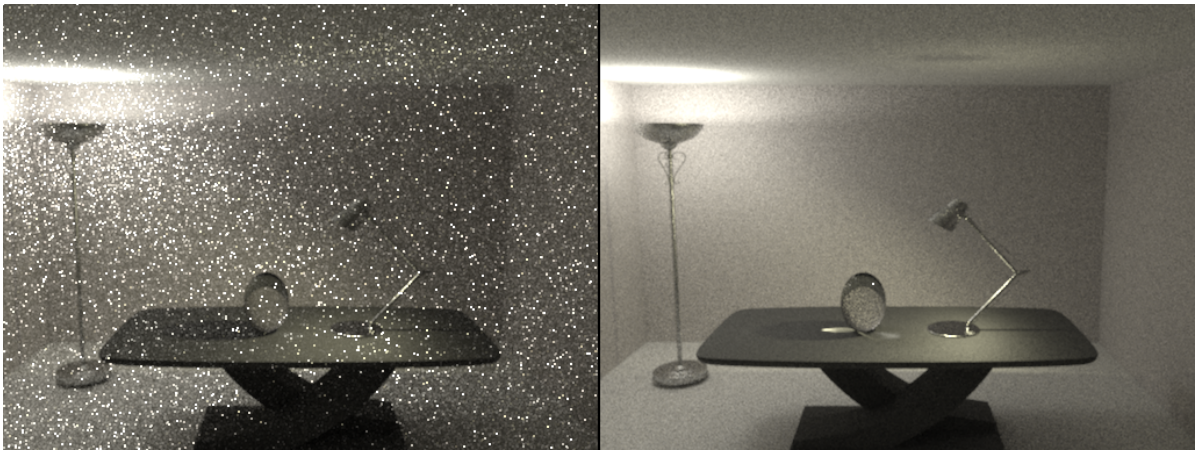


Figure 2.4: Comparison of noise using straightforward path tracing and bidirectional path tracing with the same time limit. The quality improvement is a result of more effective sampling.

generated from the light and these two paths are connected after a predetermined number of bounces. An example of the difference this can make is shown in Figure 2.4. However, even with these and many related techniques, it can take hours to reduce noise to acceptable levels [21].

2.3. Image Reconstruction

Another way to tackle the problem is to consider it from a signal processing perspective. The insight is that noisy images produced by Monte Carlo path tracing can be denoised with similar techniques as normal images [27]. What sets computer generated images apart from regular images is that there is a significant amount of auxiliary information available, as seen in Figure 2.5. These features can often be used to clearly distinguish edges and texture details from noise, and to decompose the lighting contribution into components like direct and indirect lighting. The availability of these features can allow for even very high levels of noise to be effectively filtered, as seen in Figure 2.6. Denoising filters have already been successfully utilised for prominent animated films like Pixar's Big Hero 6 [46].

However, the problems with these image filters are twofold. Firstly, the challenge of noise filtering is to distinguish noise from high contrast details that should be preserved instead of smoothed. Advanced filters like the

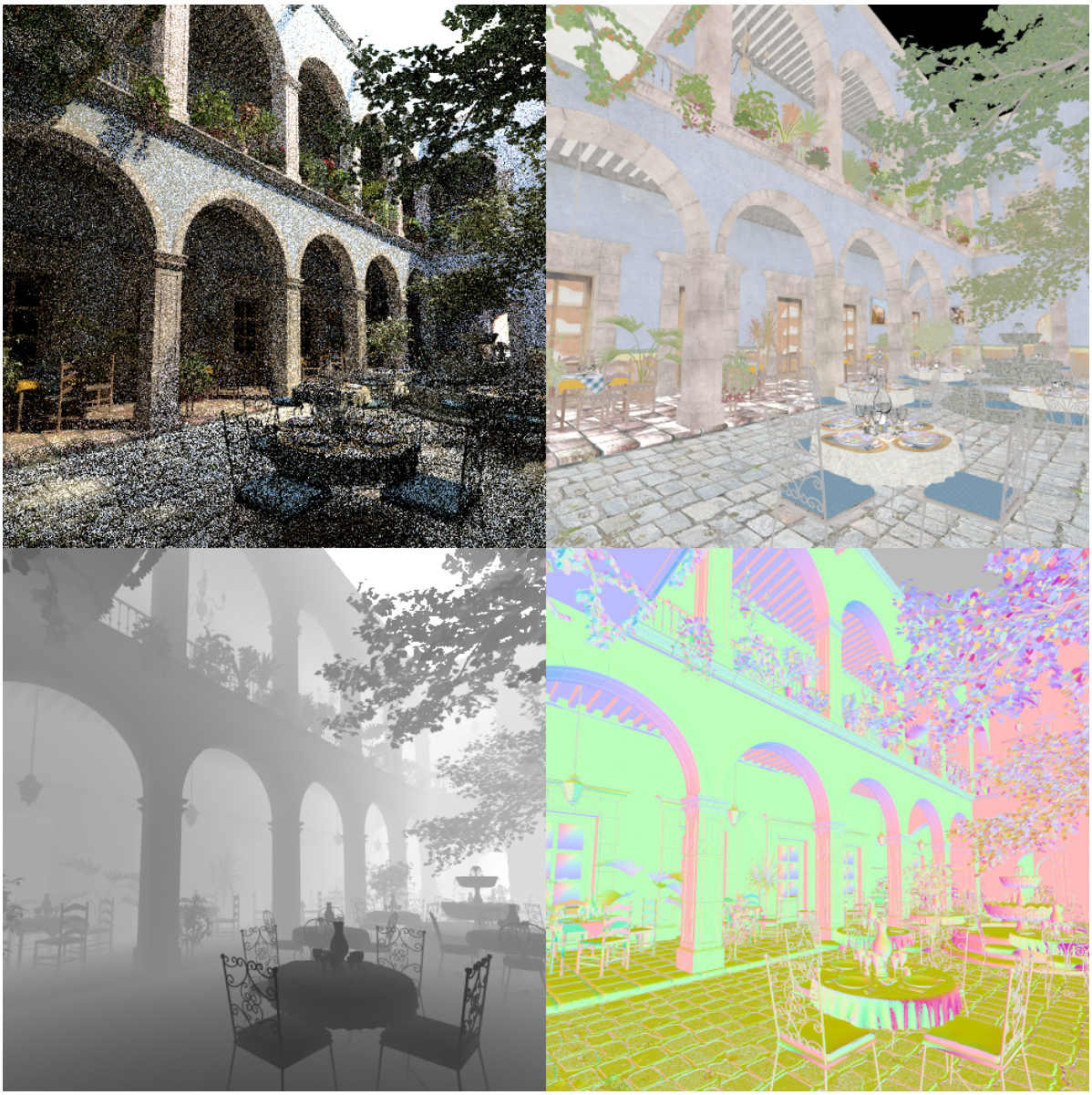


Figure 2.5: Monte Carlo path tracers can output auxiliary features like depth and surface normals along with the output image.

Figure 2.6: Monte Carlo images can be filtered very effectively using image denoising filters that have access to auxiliary scene features [52].

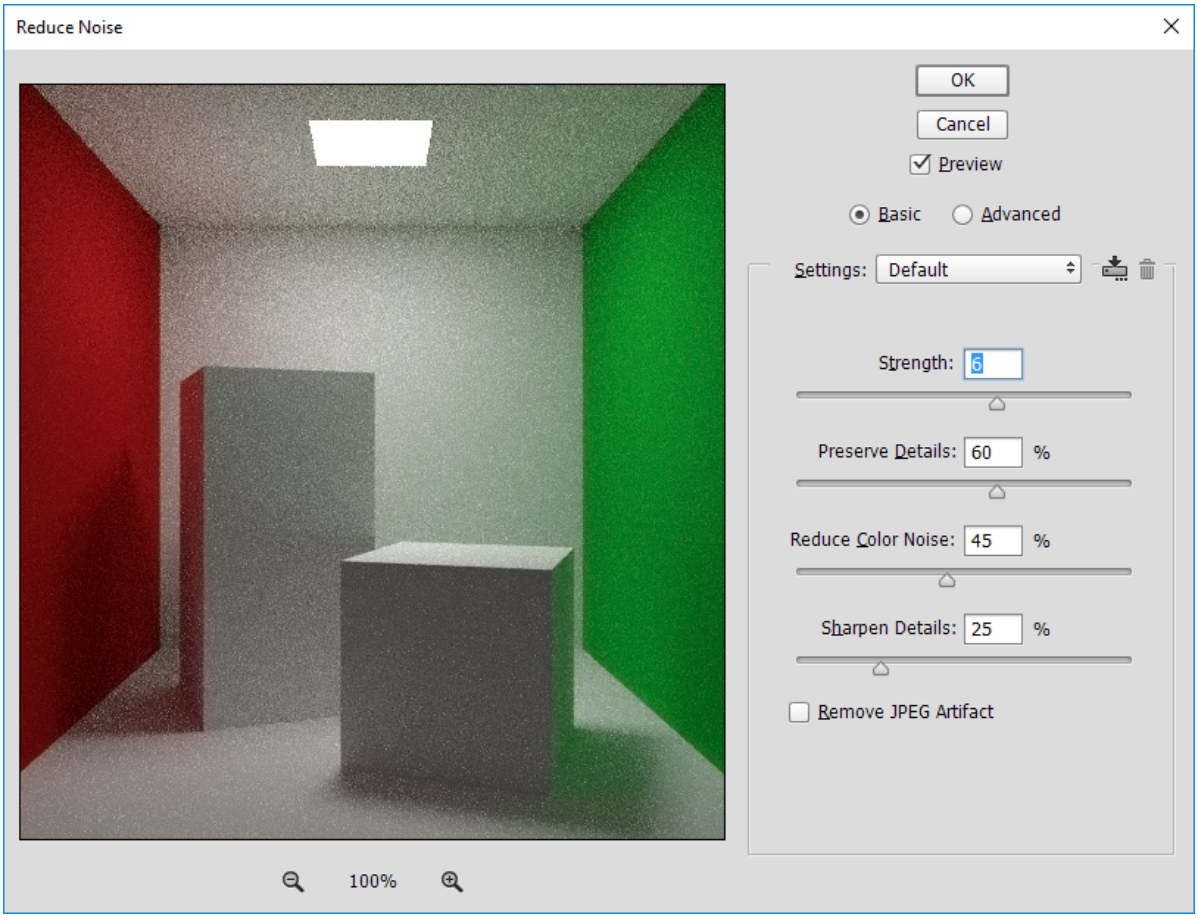


Figure 2.7: Slider-based interface for configuration of a noise reduction filter in Adobe Photoshop CC 2015.

BM3D filter [14] have a high dimensional parameter space with abstract threshold parameters and smoothing kernel size that need to be tuned per image. The effect of these parameter values on the output is often not very obvious even to the original author. All too often these filters are introduced with magic constants that happen to work well for *most* images. Secondly, Monte Carlo images distinguish themselves from normal images by having varying standard deviation throughout the image, because each optical effect has its own noise characteristics. Existing filters rely on estimators like SURE [34] to estimate noise variance in small windows to locally tune filter parameters, but these concepts assume that the law of large numbers applies rather than the small number of Monte Carlo samples that are actually available. As a result of this they have trouble distinguishing high frequency textures from noise. Even state-of-the-art approaches that use a sparse ground truth pixels with a very high amount of samples still have trouble with outliers that result in blurry spots [7].

2.4. Manual Tuning

In cases where insufficient noise statistics are available, filter parameters may even need to be tuned by hand. This is a common approach for noise removal for regular images that only have camera sensor noise with a uniform standard deviation across the image. These images do not have detailed statistics like sample histograms [15], surface albedo or depth, although it is possible to use certain tricks to acquire these [17]. This noise removal is often part of a standard photo post-processing workflow and most commonly involves slider-based interface with a preview as pictured in Figure 2.7.

The advantage of this manual workflow is that humans have a ground truth image in mind while tuning the filter that they can use to accurately judge the result. However, tuning every slider to find the right combination to produce the desired result is an aggravating process. Summarizing the high dimensional parameter

spaces of complex denoising filters with a few sliders with enigmatic names like *Strength* and *Preserve details* leaves users guessing as to what effect they have.

2.5. New Filtering Workflow

We use these insights to propose a recipe for the ideal workflow for filtering Monte Carlo images with the following characteristics:

- **Use artist insight**: The ground truth image in the imagination of the artist should be taken advantage of to converge to the desired result.
- **Offload tedious work**: Users should not have do interact with the user interface to do anything that could be automated.
- **Black-box filter approach**: Users without an image processing background should be able to take advantage of any type of filter, no matter how complex and regardless of how they work internally.
- **Extensibility**: Newly developed image filters should be effortlessly integrable in the existing workflow.
- **Ease of adaption**: The workflow should easily integrate into existing software and post-processing work.

We first propose a workflow and user interface to fulfill these principles, and then we show through a case study that it allows novice users to quickly and effectively use complex Monte Carlo filters to denoise images.

3

Related Work

Monte Carlo images have some unique advantages compared to regular images in the sense that path tracers can output a significant amount of extra data, as seen in Figure 2.5, which can be used to guide image filters [36]. Statistical details like sample variance per pixel are also often available and many algorithms have been developed to use this data to automatically tune filters. We will briefly cover some of the most significant developments. A full overview can be found in a recent survey [59].

3.1. Variance Estimation

Stein's unbiased risk estimate (SURE) [53] has been successfully used to estimate the noise standard deviation for application of the non-local means filter [56] and cross-bilateral filter [34]. However, these methods are based on dubious application of the law of large numbers to a small number of Monte Carlo samples in small windows across the image. This results in estimation errors when the number of samples is low and causes high frequency scene details to be smoothed. Wavelets have also been used to estimate noise variance [43], but these approaches result in ringing artifacts when the number of samples is low. Another wavelet based metric to estimate variance is the median of absolute differences (MAD) [16] to tune general image denoising filters [27], but the use of general image denoising filters prohibits the use of auxiliary information. However, even filters that can use auxiliary information tend to heavily rely on the auxiliary information to be noise free, which is not the case for scenes with in-camera effects like motion blur and depth-of-field. It is also possible to use differences between filters as a proxy for standard deviation [49], but approach underestimates variance in areas where light is hard to reach.

All of these approaches share the common problem that they require filters to be uniquely characterized by a standard deviation, so they are not truly general. They also all heavily rely on their metric for noise standard deviation being accurate and this is often not true for images with a relatively low number of samples. Rather than relying on noise variance estimates, it is also possible to create a sparse ground truth image [7]. The idea is to distribute a large fraction of the sample budget to a small number of pixels in the image to approximate the ground truth value and use these pixels as an oracle for choosing the best filter from the filter bank. This method still heavily relies on these ground truth estimates being accurate, which may not be true for complex scenes, but it does provide a black-box filter choice metric that only relies on the output images from filters. There has been a proposal to use machine learning [28] to generate an appropriate filter kernel from the input, but it needs to be explicitly trained on different types of scenes and effects.

3.2. Parameter Tuning Interfaces

This black-box approach to filter choice is also used in manual post-processing workflows. Artists use sliders in software like Photoshop (Figure 2.7) to configure the parameters of filters and compare the live preview to the ground truth in their imagination to judge the quality of the result. Although this approach ultimately results in exactly the image that the artist desires, it requires tedious interaction with the slider-based interface

Figure 3.1: Design gallery interface for placement of lights in a scene with a hierarchy of example placements represented by render previews. The variance of the examples decreases in the lower levels of the hierarchy to converge to the desired outcome.

to find the result that is just right. There have been proposals to augment this type of interface with previews that show the effect on the outcome for movement of each individual slider like Photoshop's Variations interface [5]. Alternatively, previews may be organized in a 2-D grid where the parameter space is reduced to two dimensions [41] [55] [30]. Although these types of interfaces take advantage of users being able to judge if one result is better than another, they are limited to exploring a small number of parameters at a time. This means that users are still required to judge which subset of the parameters most significantly affect the outcome. We believe that this explains why these interfaces were found to be less effective than sliders in a case study.

Instead, we draw inspiration from the original concept of design galleries [35] that these *image-navigation* interfaces are based on. The proposed workflow consists of two stages. In the first stage the parameter space is randomly sampled to build a large bank of example outcomes, a generalized version of the filter bank concept in Monte Carlo filtering approaches. These examples are then organized into a hierarchy of clusters by an appropriately chosen distance metric. Users can then navigate to the desired outcome by recursively selecting the most preferred cluster center, as seen in Figure 3.1.

The previews in the upper levels of the hierarchy show a representative overview of the entire parameter space by maximizing outcome variance. This allows users to quickly converge to the desired outcome regardless of the parameters that lead to it and prevents them from getting stuck in a local optimum. The original research shows that this paradigm is applicable to a wide range of problems with high dimensional parameter spaces and large parameter intervals, and we propose to use it for Monte Carlo filtering.

While this approach does not have the shortcomings of the 2-D grid based interfaces, it does have some problems of its own. If the outcomes in the example bank are highly variant, the images that are chosen as cluster centers in the upper levels of the hierarchy may not be representative of all the examples that are clustered within. This leads to frustration when users know that there is an outcome that they like, but they

are unsure of which cluster contains it. This is a problem with hierarchical image navigation in general [32], but can be alleviated by using soft clustering algorithms [24]. The idea is that some images may be assigned to multiple clusters if they are not in close proximity to any one cluster center.

3.3. Filter Generation and Selection

Another possible source of frustration is the inability for the user to find any outcome that looks acceptable. To return to the problem of Monte Carlo filtering specifically, this may happen because different regions within the image demand different optimal filter choices, whereas the aforementioned interfaces only allow a global choice of outcome. Our proposed interface solves this by allowing users to locally optimize choice of outcome. It is also possible that there are regions within the image for which an acceptable filter exists, but it is not part of the filter bank as a result of poor parameter space coverage. The original design galleries paper uses uniform random sampling for the parameter space to generate example outcomes, which does not guarantee exhaustive exploration of the whole parameter space. Different sampling methods like stratified sampling [50] can provide these guarantees to possibly create a more representative bank of outcomes. Latin hypercube sampling can be used to sample the strata of high dimensional parameter spaces [37].

3.4. Case Study

The effectiveness of image based parameter tuning interface has previously been evaluated with a case study [30]. Our case study will use a similar approach to compare interfaces. Participants will be shown a target image and given a fixed amount of time to use an interface to approximate this target image as much as possible. The subjects will be novice users who are not very familiar with image processing or computer graphics at all. Participants fill in a questionnaire at the end to describe their workflow and experience. For interpretation of the qualitative results from the questionnaire, we use common methodologies for ensuring reliability and scoping the research questions [8].

4

Overview

Our main contribution is a complete workflow for letting artists optimize noise filters for Monte Carlo images. Our contributions towards that workflow are:

- A new interface for locally painting filters
- An image-navigation based interface for choosing the locally optimal noise filter to apply

We will first introduce our proposal for a new local noise filter application workflow in section 5. The workflow allows users to paint filter parameters to images with per-pixel granularity. The tools and interface that facilitate such a workflow are covered and we discuss how the different filter choices across the images can be smoothly blended using various compositing techniques.

Although our parameter painting workflow provides users with a new way to apply a chosen filter to regions of an image, it does not help them select the right filter. In section 6 we move onto our proposal for an image-navigation based approach for noise filter choice. Instead of providing users with sliders to manually tune filter parameters, we provide them with example images of results that they can achieve with different noise filters and parameters. We show how soft clustering can be used to let users navigate to the most desired filter result.

While initial noise filtering will result in substantial improvements in image quality, the fine tuning of filters requires users to gain a more detailed insight into properties of the image and the remaining noise. In section 7 we introduce various visualizations to guide users to areas of the image that could be improved.

As discussed in section 3, many noise filters heavily rely on noise free auxiliary features, but in-camera effects like motion blur can introduce noise in these buffers as well (Figure 8.1). We show that our workflow could be extended to include filtering of the auxiliary features as a preprocessing step to improve the performance of filters in section 8.

Next, we cover the details of our implementation of the workflow in section 9. We show the design decisions that have to be made to offer an interactive and satisfying user experience in practice. The architecture of our software, implementation of the UI and choice of filters to populate the filter bank are also discussed in this section.

Finally, we introduce the design of a case study in section 10 to test the hypothesis that our workflow is superior to existing slider-based filter tuning approaches and discuss the results of this case study.

5

Parameter Painting Workflow

In this section we introduce our new paradigm for local filter selection and blending that involves the *painting* of filters.

5.1. Traditional Workflow

Users who work with photo manipulation software to denoise images currently start by tuning a single filter that works well for most of the image. They may then add more layers with different filters and blend these together to apply the most appropriate filter to every region of the image. Managing the layer order, filter choices and masks quickly becomes unwieldy, meaning that users generally constrain themselves to three layers or fewer.

5.2. Painting Parameters

Rather than first rendering filters to separate layers and then painting layer masks to make the right choice for every pixel, we propose to unify the process. We introduce an interface that allows users to directly and locally apply the right filter by painting the choice of filter and parameters directly into the image through a *parameter map*. An example is shown in Figure 5.1 where a Gaussian filter is applied to an image with a map controlling the standard deviation.

There is a set of parameter maps that describe the chosen filter and exact parameter choices with a per-pixel granularity. Analogues to the brush color selection in photo manipulation software, our interface allows users to assign a filter and parameter choice to their virtual brush and start applying it to the image by painting. Users are directly painting the parameter maps and these modifications are used to render an updated composite image. An example of such an interface is illustrated in Figure 5.2. This abstraction of parameter maps allows for interesting new tools. For example, users could apply an additional blur filter to the parameter maps themselves to smoothly blend between different parameter values. The viability of such features depends on a continuous mapping between filter parameters and output, however, like the spatial sigma (σ_s) parameter of the Gaussian kernel.

5.3. Iterative Application and Blending

Although direct painting of parameter maps allows users to quickly apply many different filters across the image without being weighted down by layers, it does not provide a direct solution for blending. Neither does it easily allow users to focus on optimization of specific areas at a time.

To solve these problems, we introduce a user interface that changes interaction into a straightforward iterative staged workflow. Users apply filters to images by moving through the following stages (visualized in Figure 5.3):

(a) Parameter map for σ

(b) Filtered image

Figure 5.1: Gradient parameter map that is used to control the standard deviation (σ) of a Gaussian filter applied to an image.

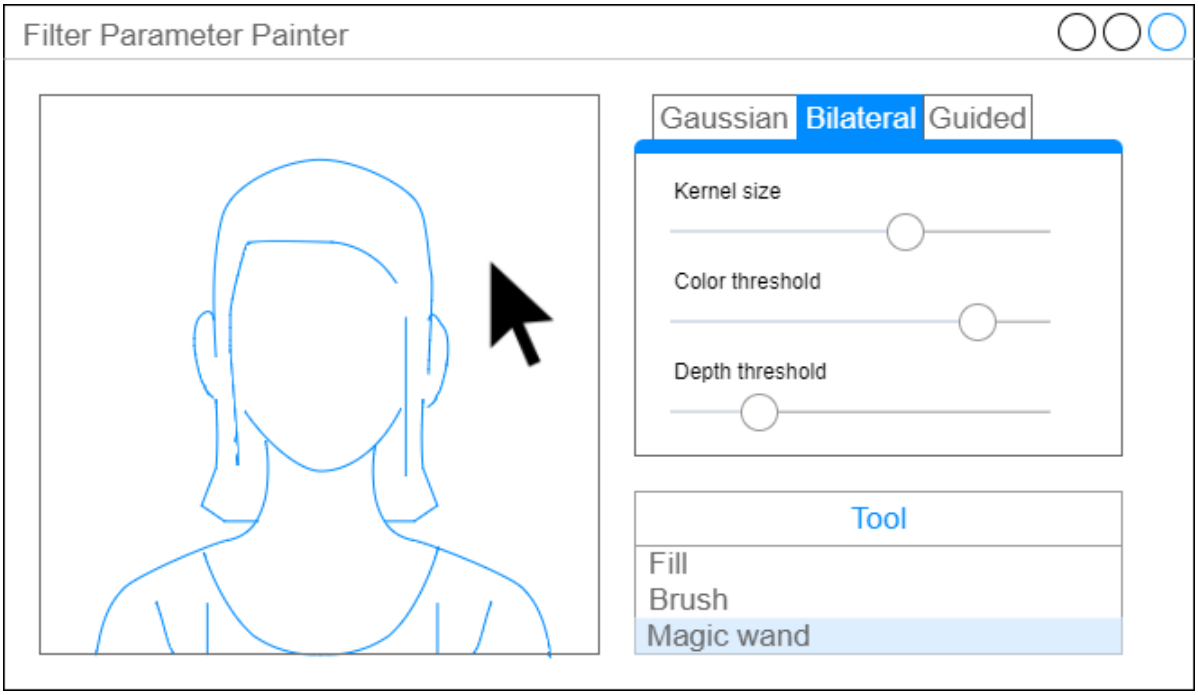
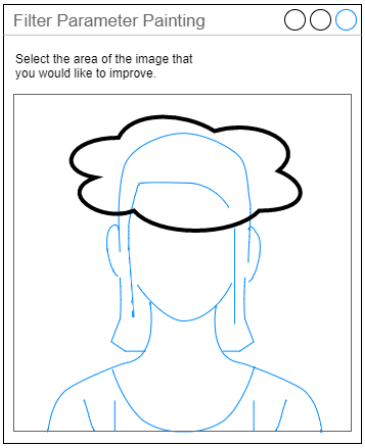
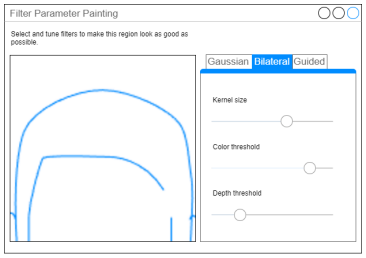


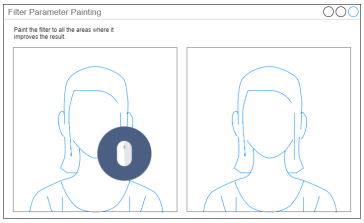
Figure 5.2: Interface for painting filter parameters directly rather than through a set of layers by replacing traditional color selection for brushes with a palette of filters and parameters.



(a) Selection of region of interest for improvement



(b) Selection of best filter for region of interest



(c) Application of filter to all improved areas

Figure 5.3: Workflow of filter selection and application streamlined into separate stages.

- **Region of interest selection:** users select a region in the image that needs improvement (Figure 5.3a).
- **Filter selection:** users find the right filter for that area using filter choice and parameter sliders while being guided with a live preview of the filter being applied to the selected area (Figure 5.3b).
- **Filter application:** users return to the full image and paint the filter to all of the areas that are improved by it (Figure 5.3c).

At first sight this workflow does not appear to be very dissimilar from the use of existing photo manipulation software, because it essentially just streamlines the process of selecting a new filter, creating a layer from it and painting a blend mask. However, we will see why the focus on a specific area of the image will make an important difference for image navigation in section 6.

After users have painted a locally optimal filter choice to each area of the image, they can enter a compositing stage where filters and parameter choices can be blended together to finish the process. We will now discuss the interface of each individual stage.

5.4. Region of Interest Selection Stage

The first stage of our image filtering workflow is the selection of an area in the image that needs improvement. The purpose of this step is to focus on local filter optimization such that users are not inhibited by the feeling that they have to make a trade-off when tuning filters. There should be a clear best choice for the area that they have selected.

Users can create a rough selection of one such area by dragging the mouse around it to create a free-form selection as seen in Figure 5.3a. If users wish to choose a baseline filter, then they can also press a button to select the whole image. Once users are satisfied with their region of interest selection, they can press a button to move to the filter tuning stage.

5.5. Filter Tuning Stage

The tuning stage provides a menu to choose a filter and sliders to configure parameters. A preview is shown next to this menu to preview what the final image would look like if the selected filter were to be applied to the region of interest, as seen in Figure 5.3b. Parameter settings are preserved when users switch to a different filter, so they can easily compare different filters to make a final decision once they've tuned every one to the best of their abilities. Once users have found an acceptable filter choice, they can move to the filter painting stage.

5.6. Filter Painting Stage

The filter painting stage, illustrated in Figure 5.3c, users can use a brush tool to paint the chosen filter and parameters to every area of the image where it improves the result. The interface lets users control the size of the brush, and allows them to add or remove the filter by holding the left or right mouse button. A large preview is shown on the side to demonstrate what the result would look like if the new filter were to be applied everywhere. Additionally, the brush tool shows a preview of the new result when a user hovers over a part of the image. After users finish applying the chosen filter to all of the regions in the image where they think it improves the output, they can exit the painting stage and return to region of interest selection to repeat the process and further refine the image.

5.7. Basic Compositing

At some point users may have painted a locally optimal filter to each part of the image, but there could be harsh edges between the different filter choices. To blend these edges together, we propose an additional feature where the edges are either blended together using the color values or through parameter interpolation. Another possibility is to allow modification of the chosen filters to reduce border gradients for a locally less optimal, but globally more optimal result. It is possible to do this by considering it as a graph cut optimization problem [7]. These modifications could be fed back to the input filter/parameter maps or they could be restrained to the composite preview image.

5.8. Brush Softness

The brush painting tool could be extended to include a softness parameter to paint weights in addition to filter and parameter choices. These weights could be used as a hint to blend filters together. This is the approach we will be using in our implementation.

The weights in this map are manipulated through the brush tool. The brush size and softness parameters are used to generate a square brush kernel with the brush size as dimensions. The kernel has non-zero values in a circle shape that touches the borders of the kernel. The softness parameter determines the radius of an inner circle within that shape where all the values are 1.0. Values outside the inner circle but within the outer circle falloff to 0.0 linearly with distance. An example of a brush with size 32 and softness of 50% is shown in Figure 5.4. When holding the left mouse button, the brush kernel weights are added to the blend buffer to increase the prominence of the new filter and holding the right mouse button subtracts the brush kernel from the blend buffer weights to bring back the existing composite.

New filters that are used to replace the current pixels in the composite are always using the original color map as input, so there's no notion of stacking filters on top of each other. Users always blend between the composite so far and a new filter, where the new filter may very well be the original noisy image. This allows users to simultaneously always move forward by further improving the image without being penalized by poor previous filter choices. We will use this approach for our implementation.

5.9. Poisson Diffusion

Another approach to blending filters is not to use a soft brush, but rather to take the filter choices of the user as a seed to produce a smoothed filter choice. The idea is that users paint the outlines of areas in the image that require different filters after which Poisson diffusion (the heat equation) is applied to optimize the filter choice in all the areas that have not been explicitly painted. This results in smooth transitions between different filter choices as seen in Figure 5.5. To blend $N > 2$ filters together, one could solve the heat equation separately for every filter that was used where the scribbles of all other filters are united as *the other filter*. The resulting weights for all filters can then be accumulated into a single map again to produce the final composite.

Our implementation as used for the example is too slow for interactive use, but we believe that it is possible to integrate Poisson diffusion as part of the live preview by using a fast solver [25].

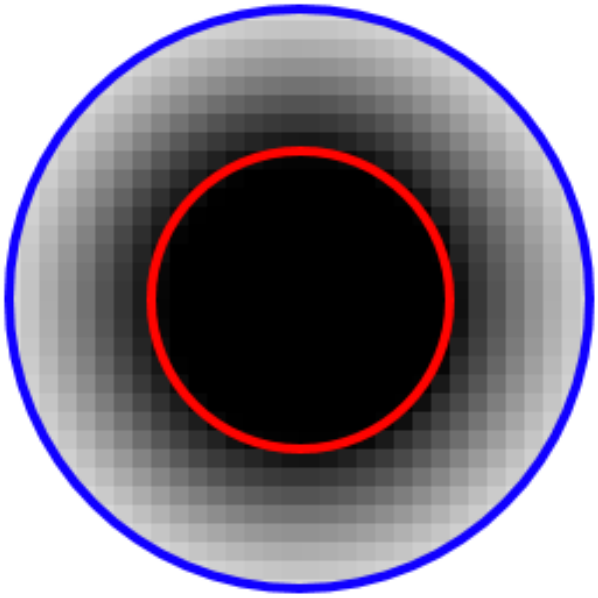
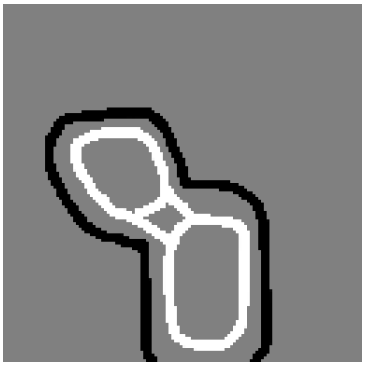
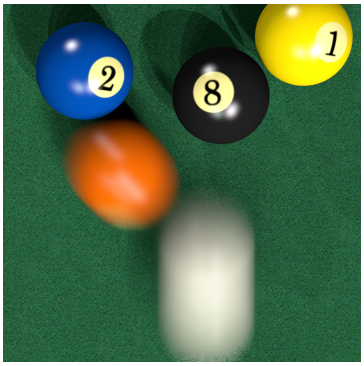


Figure 5.4: Example of a brush kernel with a size of 32 and a softness of 50%. The inner circle has a radius of 16 and contains values 1.0. The values linearly falloff to 0.0 between the circles.



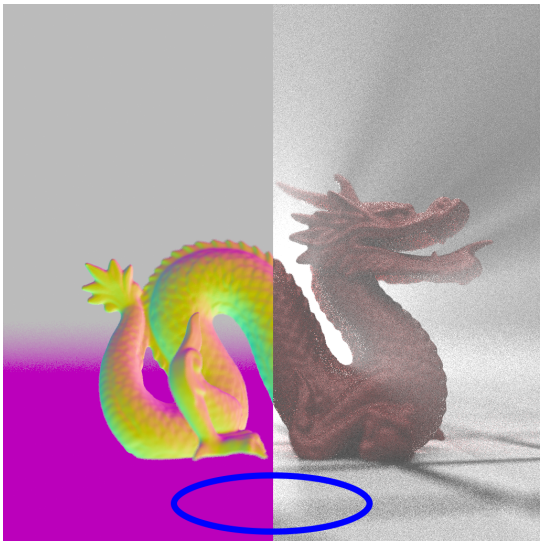
(a) Filter choices painted by user. Black represents a filter that preserves original details and white a large Gaussian blur filter. Grey is a neutral color that represents a lack of explicit filter choice.

(b) Filter blending weights after optimizing Poisson diffusion given the user filter choices.



(c) Final result of POOLBALL scene being filtered using blend map.

Figure 5.5: Deriving a smooth filter choice map from user input through Poisson diffusion.



(a) Seed region selection

(b) Selection resulting from flood fill

Figure 5.6: Magic wand selection of floor in DRAGONFOG scene.

5.10. Multi-Channel Magic Wand

A brush is just one of the possible painting tools for filter/parameter application. We have also experimented with a multi-channel version of the classic magic wand selection tool to automatically paint a homogeneous area with the new filter. The idea is to have users roughly select an area that they want to paint by selecting a seed region. The program will then find the maximum pixel gradients in the color and auxiliary maps to do a flood fill from this seed region. An example of cases where this works well is wall and floor selection, by limiting the gradient in the normal map, as demonstrated in Figure 5.6. However, we found that is not effective in scenes with more complex details like the leaves in the SANMIGUEL20 scene (Figure 10.2), so we chose not to rely on it for our implementation.

6

Image Navigation

Although our parameter painting workflow intends to improve the experience of applying and compositing locally optimal filter choices, it does not do much for making filter choice itself easier. Users are required to try out all the filters and move sliders around to find the best match. Image navigation is our proposed new paradigm for filter parameter exploration that tackles this problem.

6.1. Design Gallery

The basic idea is to guide users towards the right parameters by generating and showing examples of possible results and letting the user pick the preferred example to converge to the best parameters. This idea was originally introduced as the design gallery paradigm [35]. We will adopt some of the nomenclature from this paper, primarily the two subproblems of designing such an interface:

- **Example selection:** Sampling the filter parameter space to build a set of examples.
- **Example arrangement:** Visualize the examples in such a way that users can easily choose the most preferred one.

We will now describe how we solve both of these subproblems to implement a design gallery for Monte Carlo reconstruction filters.

6.2. Example Selection

Examples are generated by applying a predefined amount of random filter and parameter choices to the input image. The set of images that results from this will from now on be referred to as *filter bank*. To summarize the whole parameter space with a representative set of examples, we propose using Latin hypercube sampling (LHS) [37]. We represent choice of the filter algorithm itself as the first dimension in the hypercube with the N remaining dimensions, where N is the maximum number of parameters across all available filters. The sampled value from the first dimension has the continuous range $[1, M]$ where M is the number of available filters and we round this value to determine which filter to use. If a filter has less parameters than the maximum amount, then we simply discard the remaining values in the sampled vector. To handle parameters with wildly different semantics and sensible value ranges, we use the range $[0, 1]$ for all parameter dimensions of the hypercube and linearly map this to the actual parameter range when a filter has been chosen.

The challenge with application of stratified sampling to exhaustively cover the parameter space is that there is an implicit assumption that there is a linear relation between parameters and output differences. This assumption does not hold even for simple multi-scale filters like the Gaussian kernel, as seen in Figure 6.1. A possible solution to better distribute the example parameters, while still not making assumptions about the semantics of the parameters, is to adaptively distribute more samples to high variance strata after the initial distribution. Given sufficiently fast filter implementations, it may also be possible to adaptively gen-

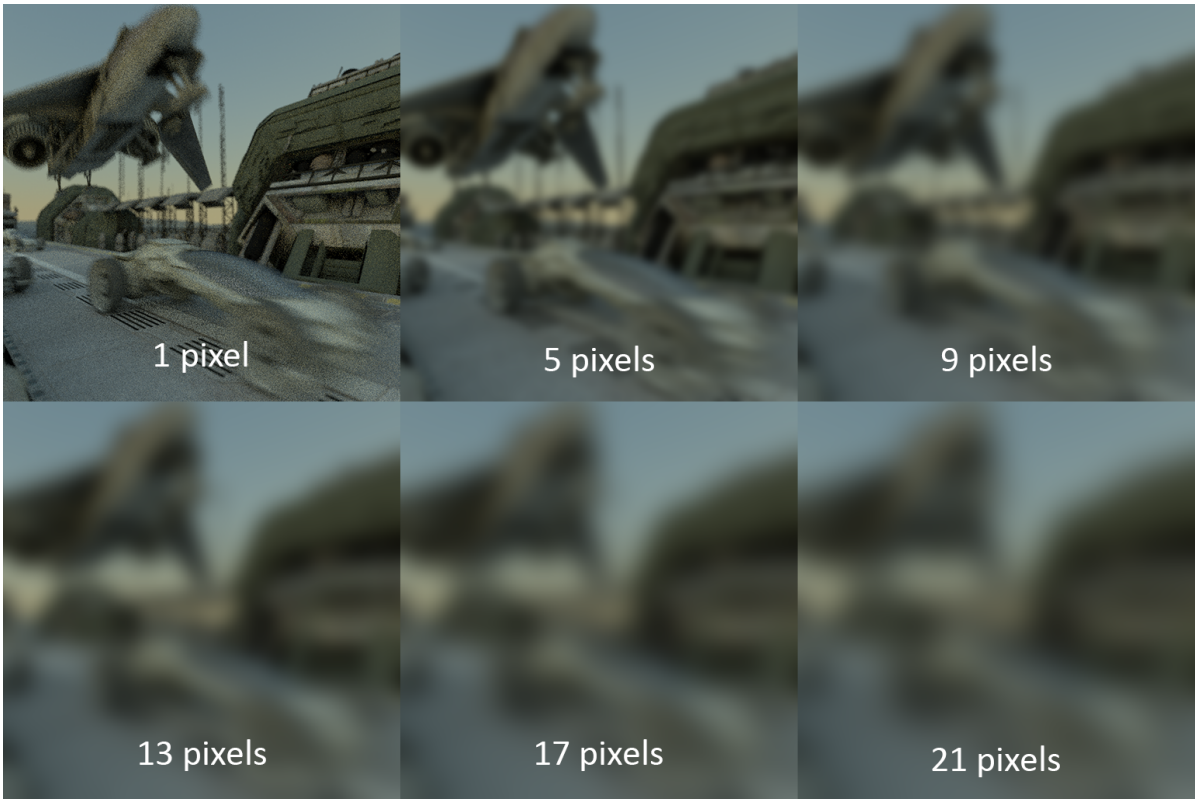


Figure 6.1: Even simple multi-scale filters like the Gaussian kernel have a non-linear relation between parameters (window size) and output differences. Note the significant variance in the lower scales and the nearly indistinguishable results in the upper scales.

erate more examples in the background based on the examples that users are most interested in, similar to interactive evolution approaches [48].

However, we instead take advantage of the idea that a large bank of mostly similar examples can be distilled into a smaller bank of more distinct examples by using a clustering algorithm. An example of this principle is shown in Figure 6.2. We will show that this approach can be united with the solution for example arrangement.

6.3. Example Arrangement

Finding an image in a large set based on a user's mental comparison image is a well known image browsing problem [24]. We use multi-level soft clustering to organize the filter example images into a tree of clusters where clustering is based on an image distance metric. Initially users are shown the cluster centers at the root node of the tree, which will be k very distinct filters. After they click the filter that they prefer, they navigate to the next level of the tree where they again see k distinct filter previews, but these new previews are closer in distance to the previously chosen preferred filter. By repeating this navigation step, users converge to their most preferred filter, hence the name *image navigation*.

It is possible for the user to accidentally pick a dead end path if the cluster root images do not properly reflect if the most desired result is contained within their subtree. The original design gallery allows users to recover from these paths through the hierarchy of images by showing all levels simultaneously, as seen in Figure 6.3a. We propose a different interface (Figure 6.3b) where only one level of images is shown at a time and the user is only allowed to move forward to combat choice anxiety [51].

The naive approach of recursively applying k-means clustering where k is the number of examples shown to the user does not prevent users from taking the wrong path and missing out on the example that they would prefer. An example of this problem is illustrated in Figure 6.4. Let the filtered image highlighted in red be the actual best fitting example in the base cluster with all 3 possible choices and the user interface only

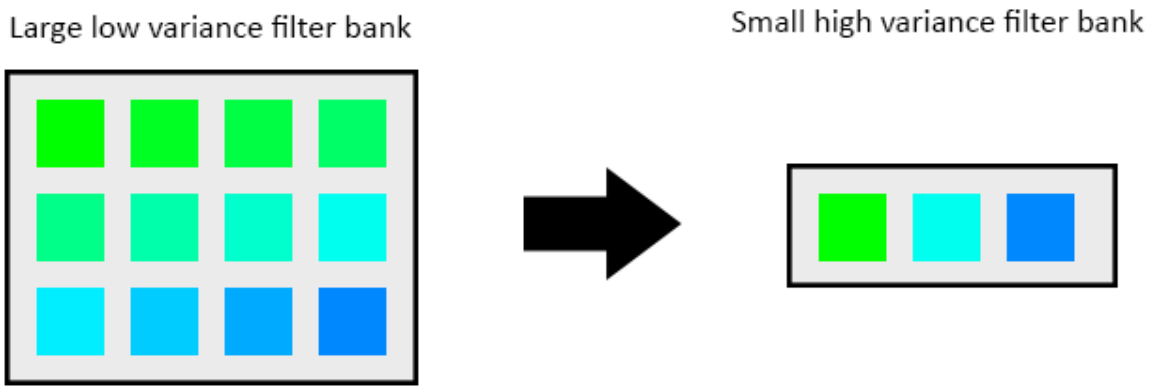
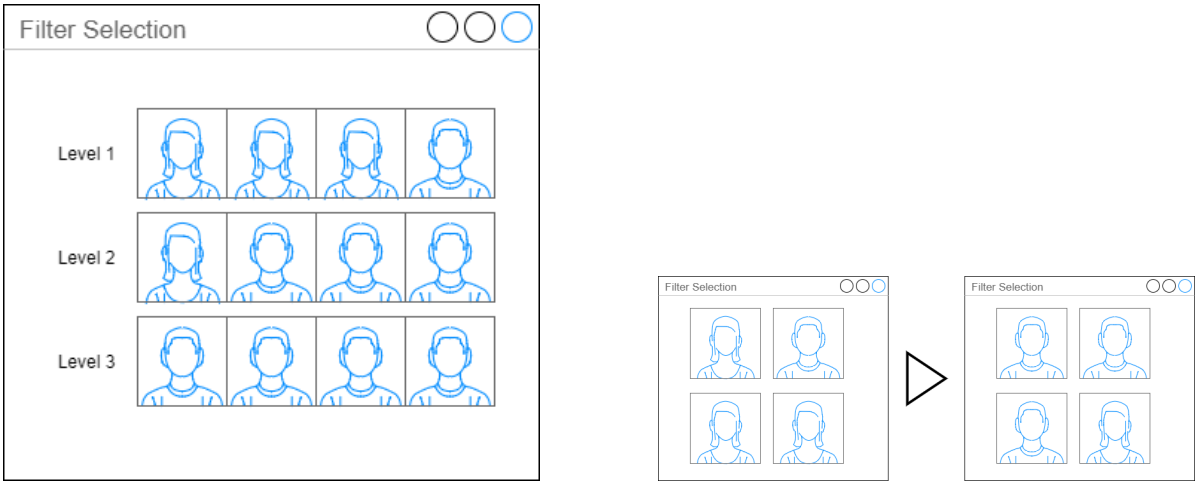


Figure 6.2: A large filter bank with a suboptimal distribution can be distilled into a smaller highly variant filter bank through cluster algorithms.



(a) Original interface (b) Our interface

Figure 6.3: Comparison of original design gallery interface and our interface where only forward navigation is possible.

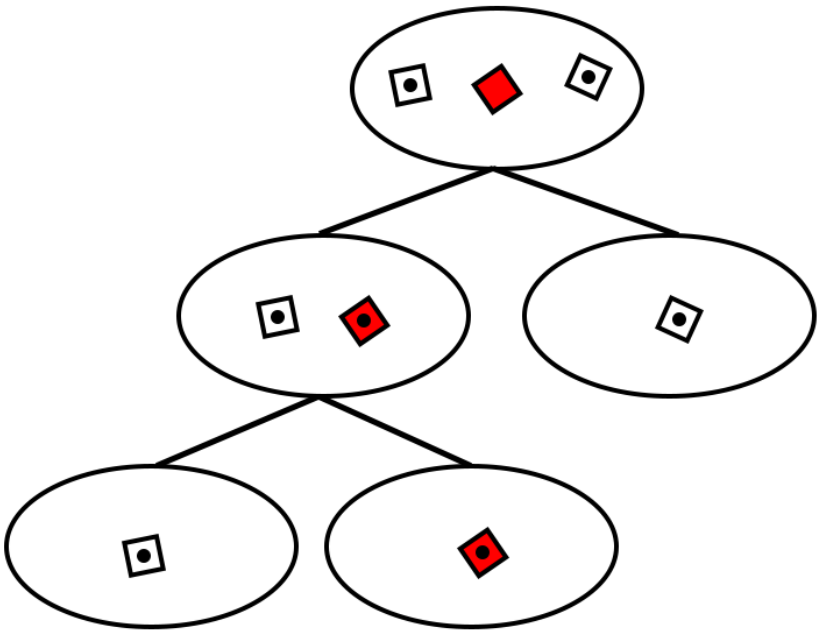


Figure 6.4: Hierarchical 2-means clustering of items. The user intents to pick the highlighted item but is unsure which cluster to navigate to pick it.

shows 2 choices at a time. Assume that the clustering algorithm will pick the left image as first cluster and the right image as second cluster, because it is the furthest away from the previous cluster choice. Both of these images may be outliers that the user dislikes, whereas the highlighted example is a middle ground choice that represents the optimal choice. The middle ground choice can only be part of one of these clusters and it is not clear to users which outlier they need to pick to navigate to the middle ground image. They may pick the right cluster and find out that they made the wrong choice and with no ability to trace their steps, they have to start the image-navigation all over again.

To alleviate this problem we instead use a soft clustering algorithm, which results in overlapping hierarchies as shown in Figure 6.5. The cluster centers are still calculated the same way, but the examples in the parent cluster are distributed differently. Instead of strictly assigning images to a single cluster, each cluster contains a fraction p of the examples in the parent cluster that are the closest to its center. The Figure shows an example of how this works when the $p = \frac{2}{3}$. By having this overlap, users are still converging to smaller clusters and eventually a single choice, but they also have some leeway to correct mistakes made in previous levels. The fraction must be configured to properly balance speed of convergence and ability to correct mistakes. Through experimentation we found that $p = \frac{1}{2}$ works well in practice with 64 examples or more.

Clustering should be based on image similarity with a metric that quantifies the effects of filtering. We believe that a metric like SSIM that quantifies structural similarity and takes human perception into account may be the optimal choice for this. Simpler metrics like MAE and MSE may lead to surprising results [57].

Examples are initially clustered by measuring similarity across the whole image, but this may not be optimal when users are only interested in optimizing a small region. Figure 6.6 shows an example where a user wants to focus on optimizing just the look of the teapot and not the background. By only calculating the similarity of examples in this region, we can increase the variance of clusters in the user's region of interest to make exploration easier.

By using soft clustering as a solution for the example arrange problem, we simultaneously accommodate the limitations of the example selection problem by creating small highly variant filter banks at every level of the hierarchy.

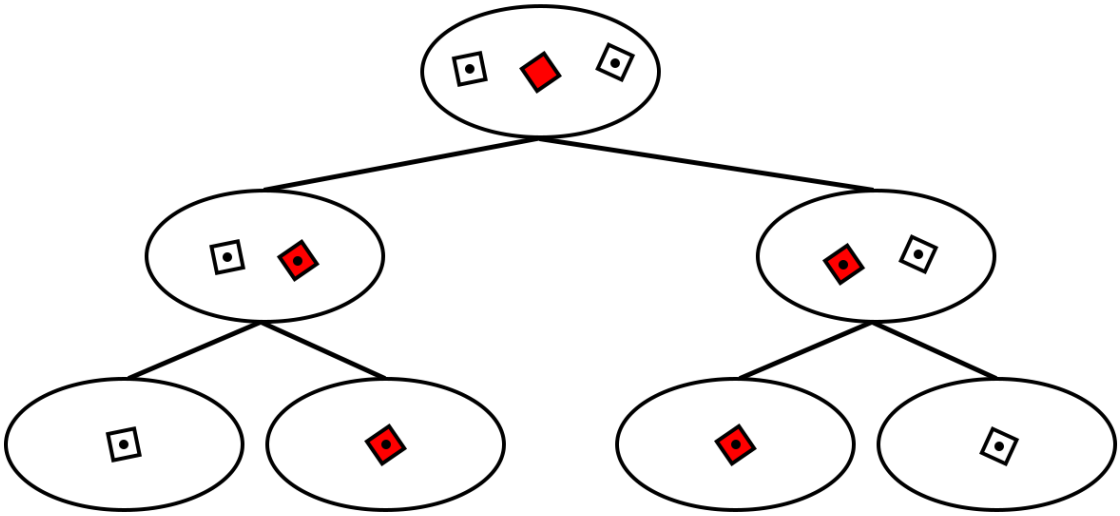
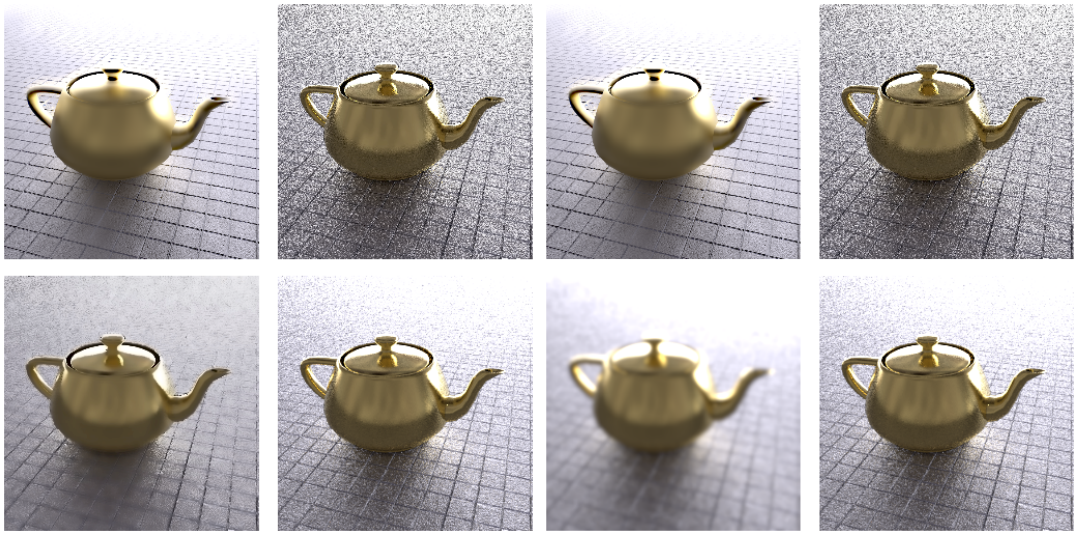


Figure 6.5: Hierarchical clustering of items with overlap. An initial wrong decision based on limited information does not prevent the user from eventually picking the item they want.



(a) Whole image clusters (b) Teapot cover clusters

Figure 6.6: Comparison of variance between clusters when using global MSE versus local MSE of the teapot cover region.

7

Visualization Aids

We provide two types of visualizations in our workflow to guide the user during the region selection and filter application stages.

To help users find regions within the image that they may want to improve, we introduce a filter variance visualization that shows how much of a difference choosing a filter can make for each pixel in the image. An example of this visualization for the Cornell box scene is shown in Figure 7.1 that shows that pixels near edges will be greatly affected by the choice of filter. The soft shadows cast by the two boxes are also highlighted, which is explained by the fact that these don't show up in any of the feature maps that are used for edge preservation and are thus significantly blurred depending on filter configuration. This variance map is generated by calculating the standard deviation of all the images in the filter bank and normalizing the value to $[0.0, 1.0]$.

In the filter application stage it may be difficult to see which regions are primarily affected by the filter choice, especially if the image is already close to the ground truth. We introduce a visualization that shows how much of a difference the new filter makes if it were applied to all the pixels in the current composite to tell the user where they should pay extra attention when applying the filter. An example of this visualization is shown in Figure 7.2. The particular filter chosen in this example is a cross-bilateral filter that removes nearly all of the noise, which means that high variance areas like shadows are primarily affected. The visualization is generated by mapping the absolute difference between the current composite and the chosen filter to the range $[0.0, 1.0]$.

We have chosen to use a simple grayscale color mapping to visualize the variance and filter changes, because human sight is most sensitive to luminance changes [40]. It may be worth experimenting with diverging color maps, but found the basic color map to be sufficiently clear.

7.1. Histogram Equalization

In an attempt to improve perceptual differences in the various visualizations, we compared normalization with histogram equalization. We expected that this method would better highlight local contrast to convey features throughout the entire image to the user. Unfortunately this would exaggerate meaningless features that would greatly reduce the signal-to-noise ratio of the visualizations. An example of the filter variation visualization with histogram equalization is shown in Figure 7.3. Although the most important features like edges are still distinguishable, there is low signal to noise ratio that fails to direct users to any features of interest, with the shadow on the wall as one notable exception. It may be possible to reduce this noise by using adaptive histogram equalization [47], although we did not test this.

Nevertheless we believe that it may be useful to further experiment with contrast adjustment techniques. We did find that histogram equalization is very useful for visualizing contrasts in non-noisy features with significant range like depth maps, with a comparison shown in Figure 7.4.

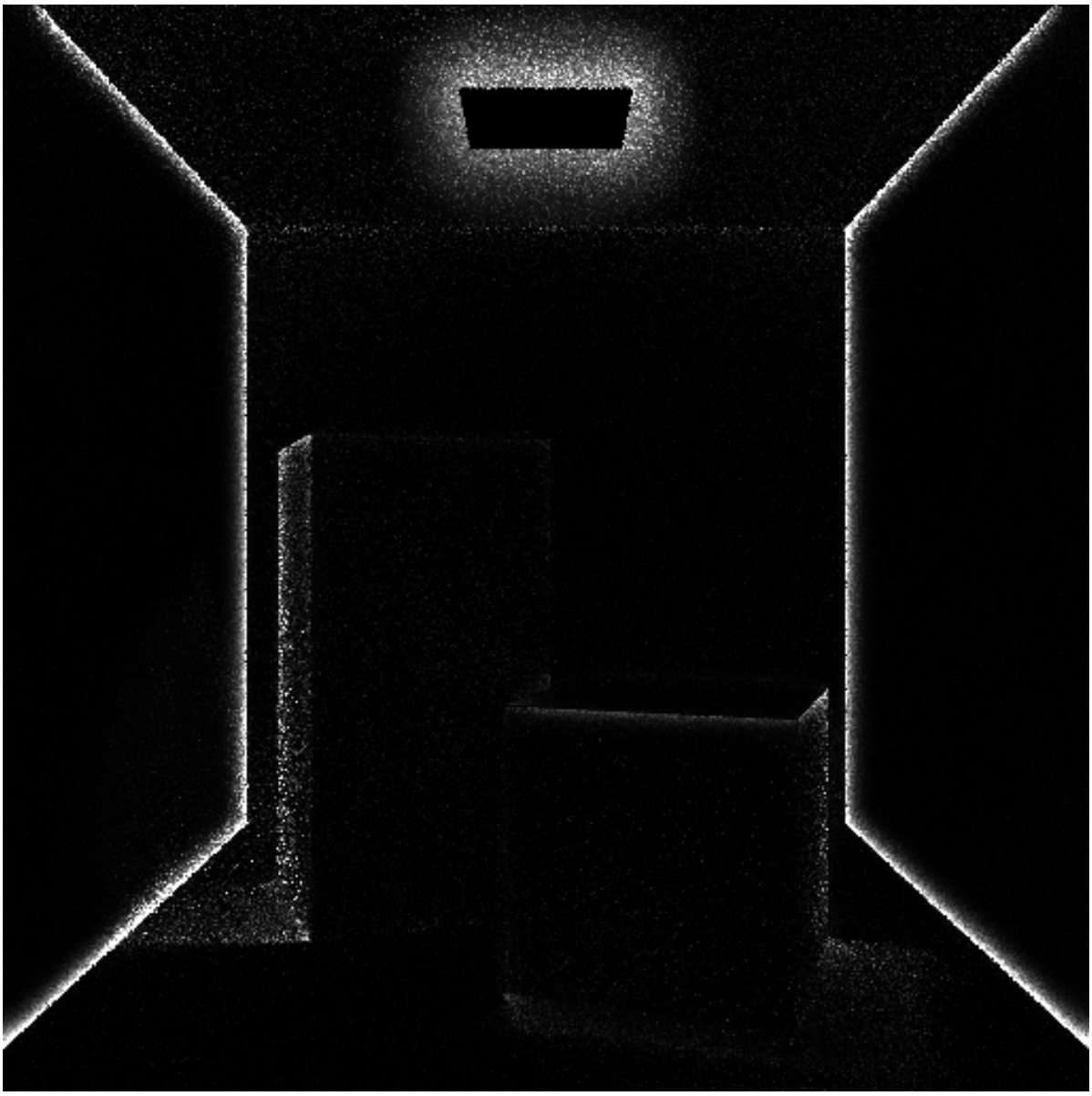


Figure 7.1: Visualization that highlights pixels in the composite image that can be significantly changed by choosing a different filter.

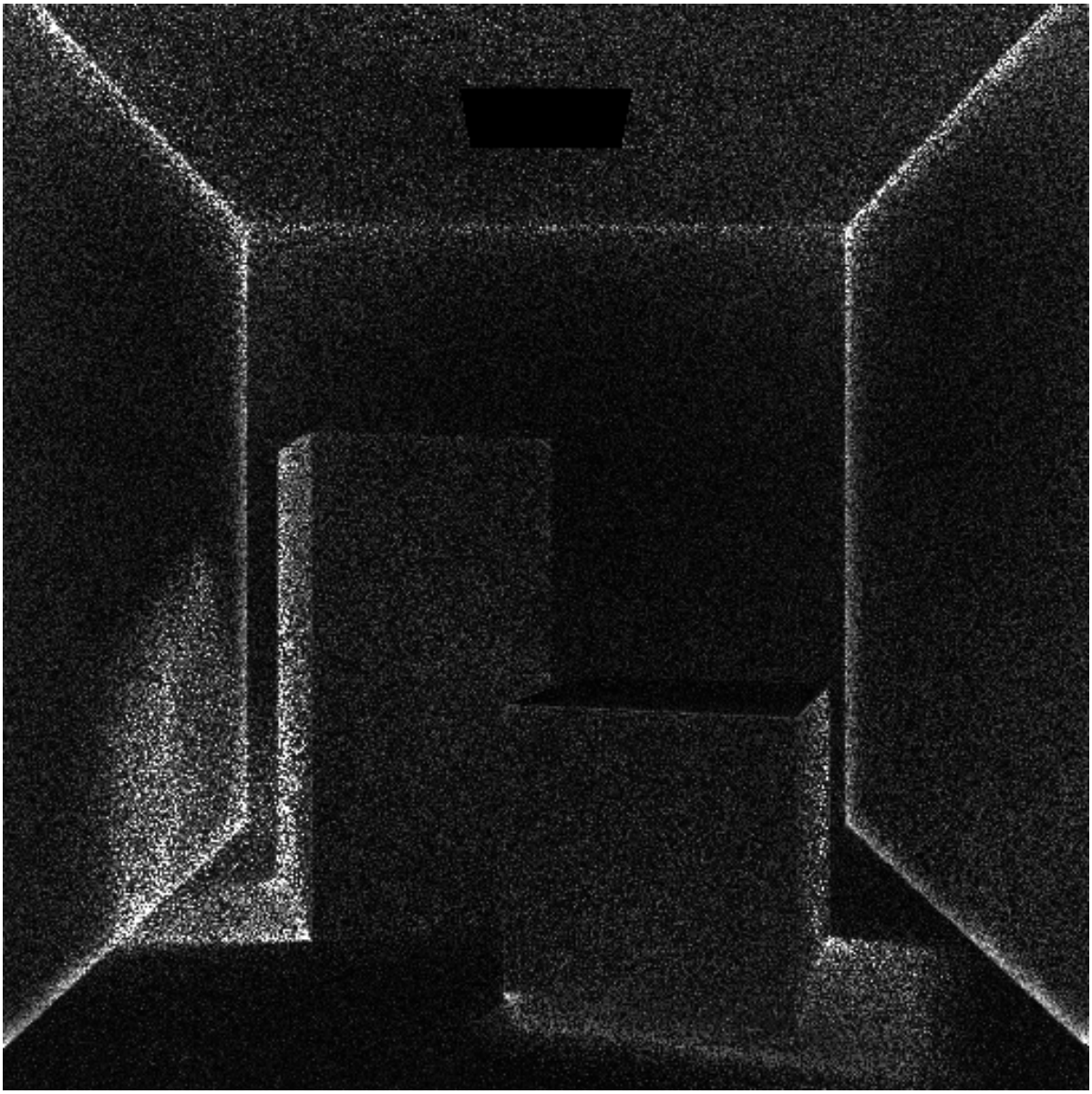


Figure 7.2: Visualization that highlights pixels in the composite image that will significantly change if the new filter choice is applied to them.

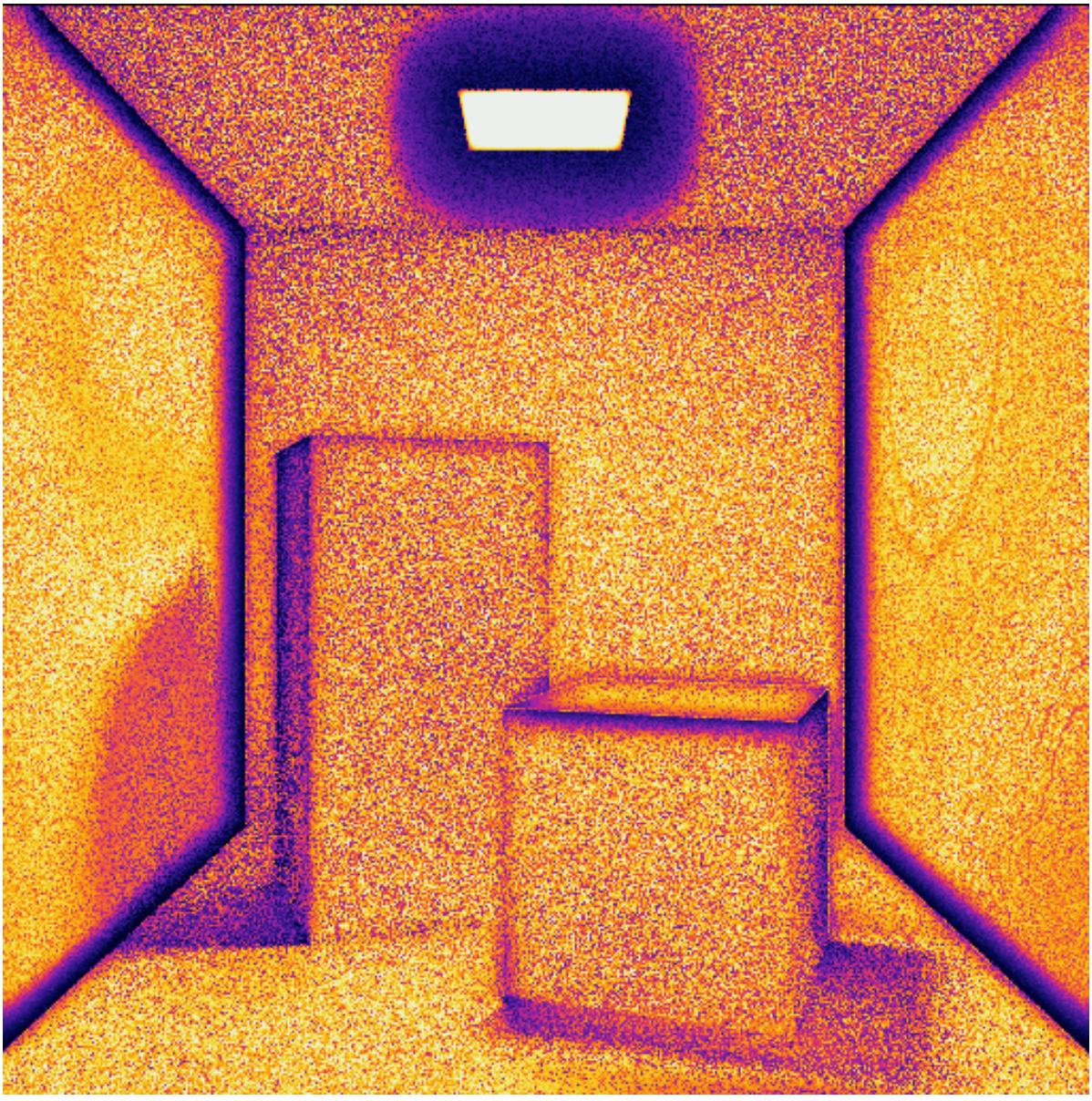
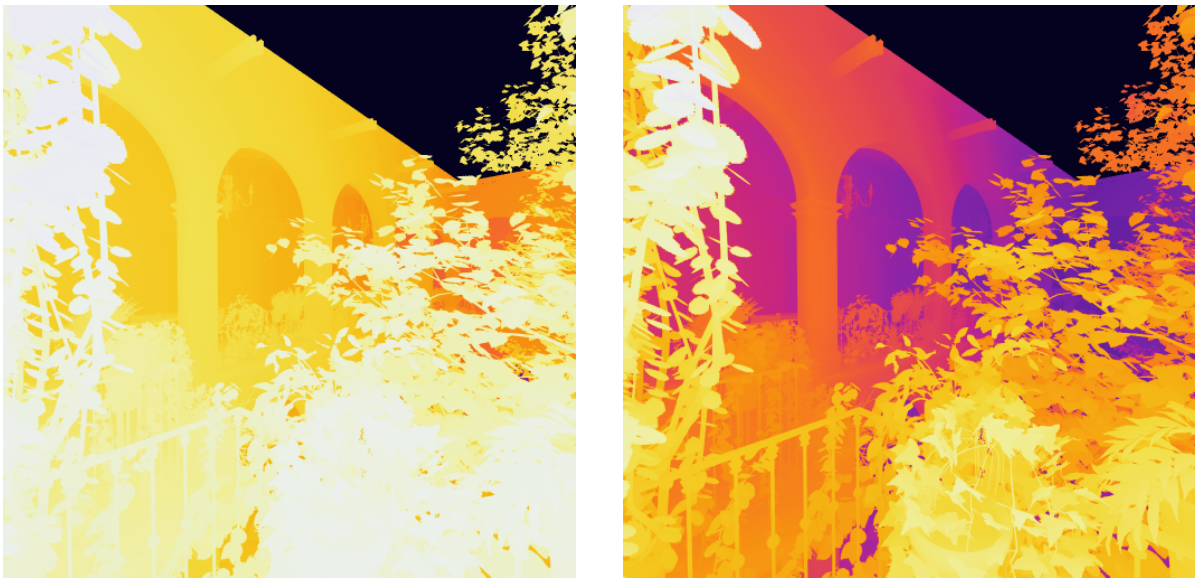


Figure 7.3: Filter variance visualized through histogram equalization and a black body ration inspired color map.



(a) Normalization

(b) Histogram equalization

Figure 7.4: Comparison of contrast enhancement methods for depth map visualization

8

Filtering Feature Maps

The vast majority of Monte Carlo filtering algorithms ultimately use the feature maps as oracle for edge detection and preservation, and they are assumed to be noise free. Unfortunately many in-camera effects like motion blur and focus blur also introduce noise in these feature maps, as seen in Figure 8.1. Solve algorithms solve this by filtering the feature maps themselves before using them as input for the main filter [39].

We believe that our workflow may be extended to include a new stage where filters can be applied to the feature maps before users enter the main workflow. We did not have time to evaluate the merit of this approach, but Figure 8.2 shows an example of an artist applying a bilateral filter to a normal buffer with motion blur noise where the sharp edges of the still pool balls are preserved and the noise of the moving balls is smoothed. We found that using this filtered buffer made it much easier to configure a cross-bilateral filter to improve the final image.

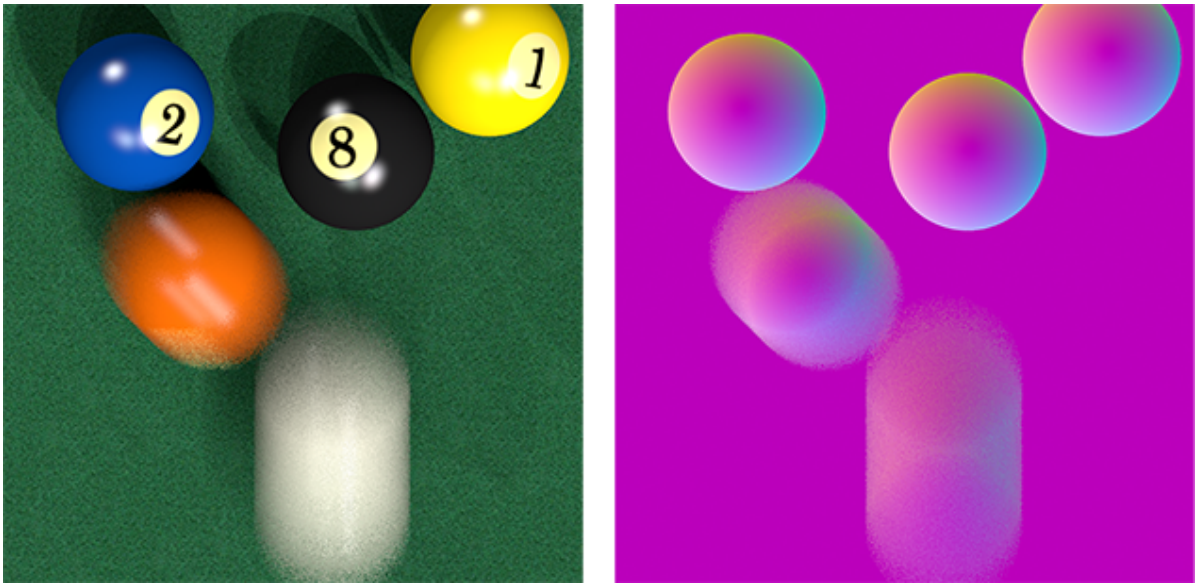


Figure 8.1: In-camera effects like motion blur introduce noise in the feature maps as well.

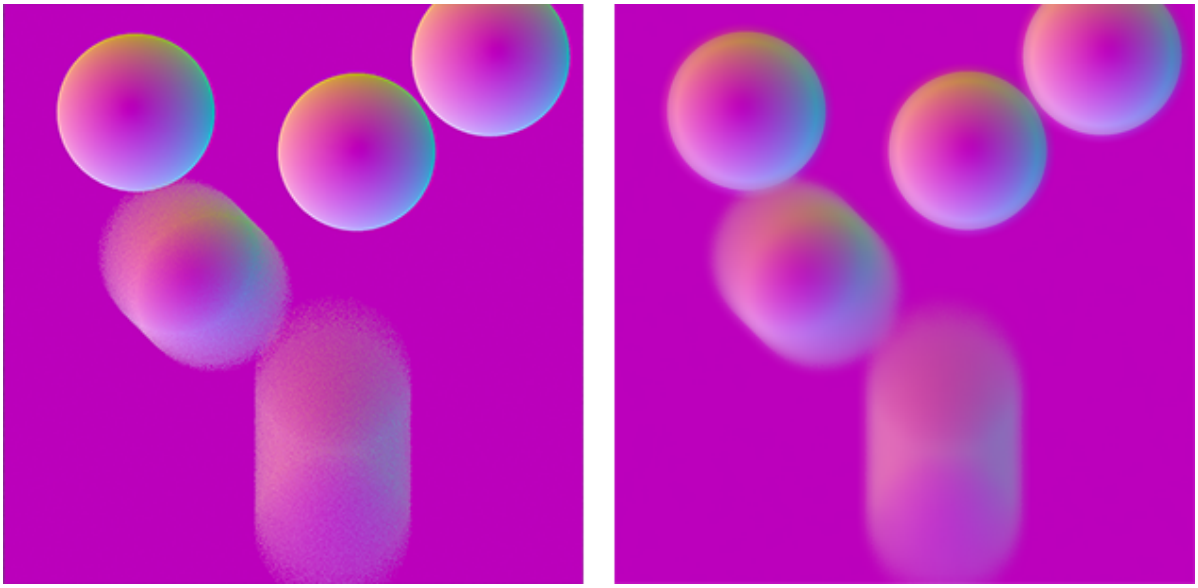


Figure 8.2: Left: Feature map containing noisy surface normals due to a motion blur effect, right: manually filtered version of the same feature map by an artist with a bilateral filter.

9 Implementation

In this section we describe how the parameter painting workflow and image navigation paradigm are put into practice.

9.1. Software Architecture

We have developed our application in C++ with the OpenCV library [9] for image processing work and the imgui library [12] for cross-platform interface design. Our application is designed to load output from Monte Carlo path tracers using the OpenEXR format, which is capable of containing color and auxiliary maps with full precision floating point values. These images are passed on to filter implementation to generate the filter bank. Each filter implementation only has to expose a list of parameters and a function that takes an input image and set of parameters to return a filtered output image. Although we currently assume that all exposed parameters are continuous, the filter implementation can easily internally map a passed parameter value to the closest correct value using its own appropriate constraints.

9.2. Interface

We have implemented the previously discussed workflow and image navigation interfaces (see Figure 9.1) where we have chosen to use MSE as clustering distance metric and to have four clusters per level. We chose to use MSE as distance metric for clustering filter examples because its performance allows us to do this in real-time. It would be interesting to compare the results with an SSIM-based metric, but we found the performance of SSIM to be prohibitive for an interactive experience. We currently compute the whole hierarchy at once when the user starts browsing which introduces a delay of a few seconds. It is possible to instead compute the subhierarchy after every level to reduce the total amount of work, but we believe that a longer initial wait results in a more satisfactory user experience compared to a short wait after every level.

Users can easily compare clusters by hovering over the previews with their mouse while seeing the live preview on the right change. Every time that users select an example as their preferred choice, they will move to a narrower cluster and pick a preferred example again. We have considered using alternative image comparison interfaces [20], but none of the concepts we found were suitable for comparing more than two images.

After the final choice they will move to the filter painting stage. It is also possible to jump from an intermediate level straight to the filter application mode if the user has no preference by pressing the *They look the same* button. This will cause the application to pick a random leaf image in the subtree that the user is currently exploring. This choice is based on the same option that patients have during an eye examination with a phoropter.

We have also implemented the slider-based UI so that we can compare its efficiency with image navigation in the case study (Figure 9.2).

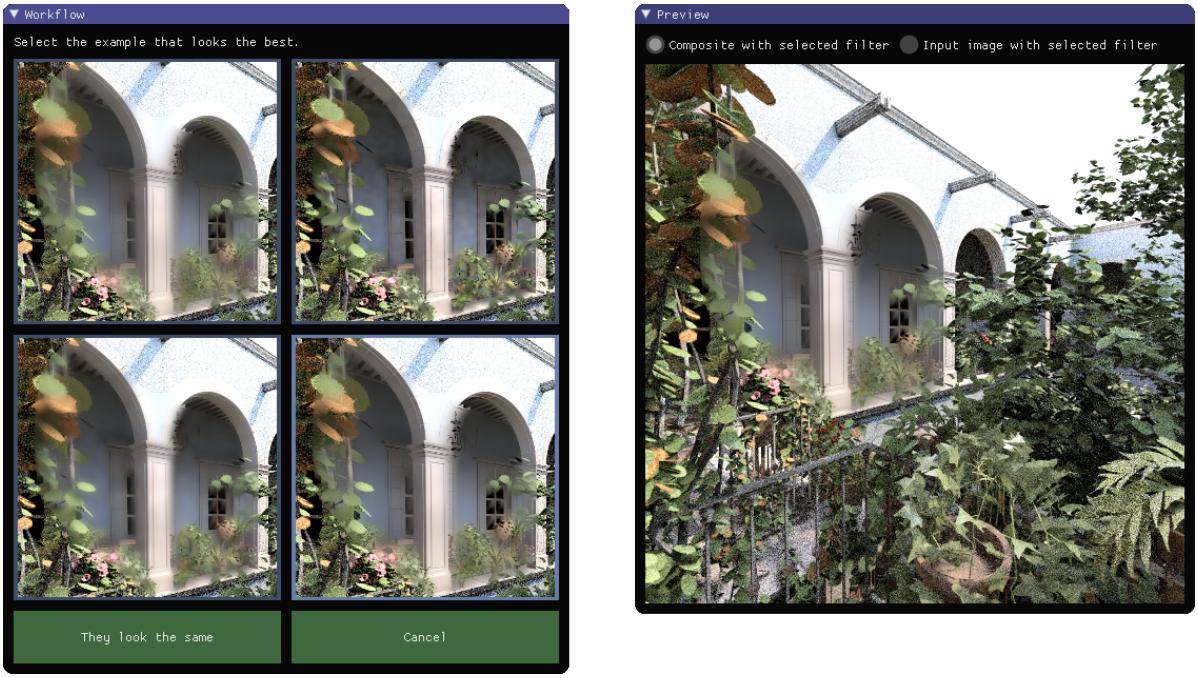


Figure 9.1: Interface for browsing a hierarchy of examples by repeatedly selecting the preferred choice out of 4 clusters.

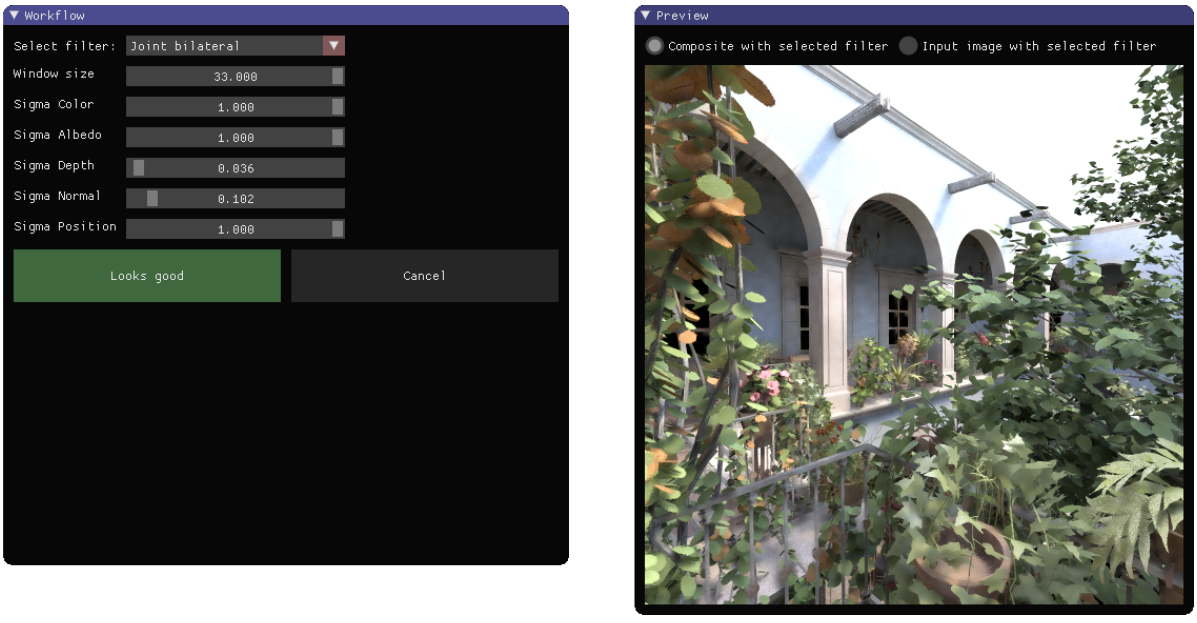


Figure 9.2: Interface for selecting and tuning a filter using sliders.

Our interface has been set up to implicitly select the whole image as initial region of interest because we found that users almost always want to start with a baseline filter. It is possible to skip through this, however. When users move to the filter painting stage, there is the question whether the filter should be pre-applied to the selected region of interest or not. We have currently chosen to not do this to encourage users that only a rough region of interest selection is necessary and precise painting comes later. It is possible that some users may find this confusing and are wondering why the filter that was there in the preview is suddenly not there yet in the painting stage. Our region of interest selection tool also currently only allows a single connected area to be selected at a time, but there is no technical reason for this to be necessary.

9.3. Filter Bank Generation

We restrained the amount of generated images to 64 on our test setup to limit the filter generation time to approximately 3 minutes. The cluster size reduction per level was set to 50% to strike a balance between tolerance for optimal choice and amount of levels a user has to navigate. To allow for offline generation of filter examples before an artist gets to work, we also implemented a simple cache system where a collection of filter examples is generated once and stored with each input image.

We have implemented the following filters to populate the filter bank:

- Gaussian blur
- Bilateral filter [54]
- Cross-bilateral filter [17]
- Guided filter [23]
- Non-local means filter [10]

The Gaussian blur, bilateral filter and cross-bilateral filter are custom GPU implementations using AMP [22]. The non-local means filter uses the OpenCV implementation and we use an open-source library for the guided filter [60].

These filters were chosen because they have fast GPU implementations that allow for real-time updates when users move sliders in the interface. For slower filters that take more than a second to execute it would be possible to generate a very large bank of examples and use the slider settings to find the closest matching example instead of generating them in real-time. We have tried this approach, but found that it only works well when the example bank is sampled in such a way that large effects caused by small slider changes are preserved. This is currently a problem with filters like the cross-bilateral filter that have a large number of threshold-like parameters that result in highly variant outputs. It is also possible that the incorrect closest example is chosen because the parameters are non-linear.

10 Evaluation

We organize a case study based on an earlier design for interface performance evaluation [30].

10.1. Goal

We seek to evaluate if our workflow allows users to effectively choose and apply appropriate Monte Carlo filters by hand. As a part of that evaluation we also intend to compare the performance of users when they use traditional slider-based interfaces versus our image navigation paradigm.

10.2. Participants

We ultimately intend to show that artists can use our workflow to efficiently apply filters to post-process still images and animated films rendered with Monte Carlo path tracing. These people tend to already be somewhat familiar with image processing filters, but not necessarily with complex Monte Carlo reconstruction filters. To test if our proposed interface really lowers the barrier, we select our case study participants to primarily consist of novice users who have little to no familiarity with image processing or photo manipulation in general. We consider users who have only used hue/saturation/brightness sliders or predefined filters for photo manipulation as having little familiarity.

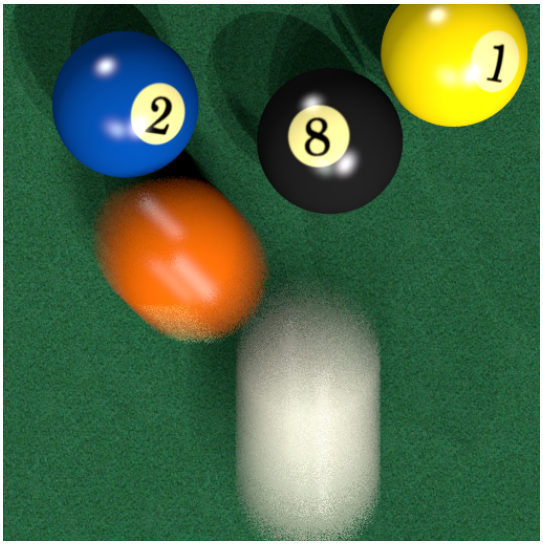
10.3. Tasks

We provide each participant with three trials where each trial consists of a noisy Monte Carlo image that they need to filter along with a type of workflow they need to use. The workflows we compare are:

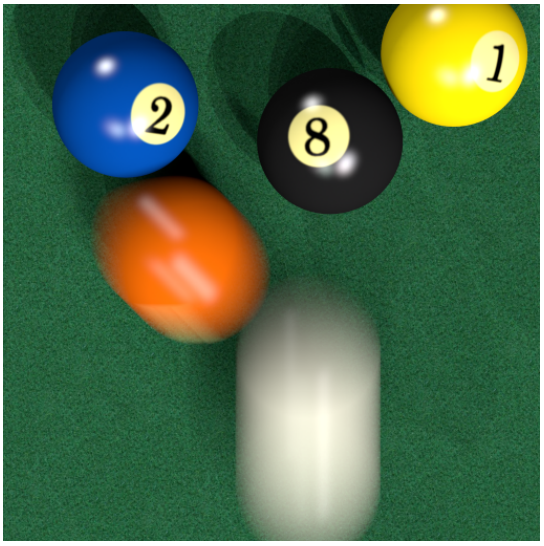
- **Sliders (global):** Participants use the slider interface to select the best filter and parameters for application on the entire image.
- **Sliders (local):** Participants use the slider interface to select filters and parameters, but may apply different filters to each part of the image through the painting interface.
- **Image navigation (local):** Participants use image navigation to select the best filter and parameters for each part of the image.

The noise in Monte Carlo images can originate from many different types of effects that result in different noise characteristics, like the overall intensity and whether they also affect auxiliary maps or not. Therefore we have chosen three images that are representative of these different types of noise:

- POOLBALL: A scene with noise from an in-camera motion blur effect and a background with very high frequency details that should be preserved. (Figure 10.1)
- SANMIGUEL20: A scene with large amounts of geometric detail and subtle texture details. (Figure 10.2)



(a) Noisy render



(b) Ground truth image

Figure 10.1: POOLBALL scene

- SHIP: A scene with an in-camera depth-of-field effect that results in foreground and background noise with a transition in intensity. (Figure 10.3)

To control for learning effect and to evaluate the performance on images with different workflows, we randomize the order of both images and workflows for every participant. The tasks are randomized in such a fashion that every image is equally often tested with each type of workflow.

In reality artists would have a mental ground truth image that they are working towards, which would mean that judging of which filter is right is inherently a subjective process. This approach would inhibit a qualitative evaluation of the performance, so we choose to explicitly provide users with the ground truth where they should approximate it as closely as possible.

10.4. Questionnaire

Participants fill in a questionnaire right after they have completed their three trials where they rate and elaborate on their experience while using each of the three different workflows. We use a 1 to 5 scale for participants to rate the difficulty of learning a workflow, difficulty of discovering the best baseline filter, difficulty of local optimization and satisfaction with the final result. They write down a few sentences about how they used the workflow and any features they missed. Although we also monitor the behavior of participants during the trials, these extra details provide us with insight about users' thoughts. Lastly, participants indicate how much previous experience they have with any type of photo manipulation, like selecting predefined filters in applications such as *Instagram*, changing hue/saturation of photos and drawing with virtual brushes.

10.5. Procedure

Participants perform the trials on a 24 inch uncalibrated TN monitor with a standard mouse and keyboard. The computer they use is outfitted with an NVIDIA GTX 760 graphics card and Intel i7-4770 processor, which is sufficient for a smooth (60 frames per second) interface and hierarchy computation within a few seconds. The office where the case study takes place is lit with regular fluorescent lighting with minimal ambient lighting. A total of 9 people participate in our initial case study, giving us 3 samples per image/workflow pair. Each trial is capped off after approximately 5 minutes and all of the intermediate composite images that are produced during that time are stored along with timestamps, so that we can also evaluate performance of participants through time.

Before participants participate in the actual trials, they are first given a tutorial image to practice each of the

(a) Noisy render

(b) Ground truth image

Figure 10.2: SANMIGUEL20 scene

(a) Noisy render

(b) Ground truth image

Figure 10.3: SHIP scene

three workflows on and may only proceed after they fully understand the interface. Participants are constantly supervised to document their behavior and to answer any small questions that they may have.

11

Results and Discussion

Our case study consisted of an initial pilot study and extended study. We will first describe the results of the pilot study as described in the previous section and then elaborate on the extension and how it came to be.

11.1. Initial Results

We first quantitatively evaluate the quality of the filtered images by comparing them to the ground truth goal image. We choose to utilize the SSIM metric for this because it takes into account the properties of human visual perception [57] and it also accounts for how well the structures in the image are preserved [58]. Both of these features allow us to evaluate how appropriate the applied smoothing is. The SSIM scores of the final composite image in each trial are aggregated by image and workflow and shown in Table 11.1. We include the SSIM of the noisy input image (*No filters*) and SSIM of the result if the best matching filter had been chosen for every pixel (*Optimal choice*).

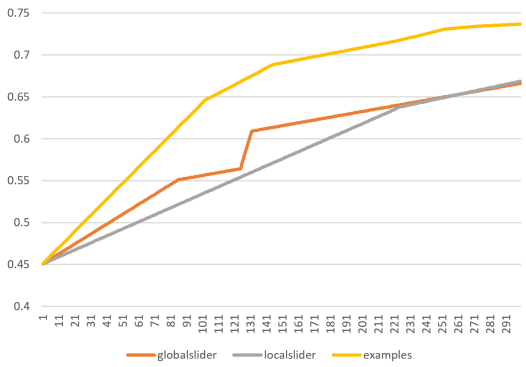
Locally optimal filter choices prove to be superior to global filter optimization, as one would expect. This may seem like a very obvious observation, but we observed that participants in the local optimization trials would often settle for a subpar baseline filter and then go on to tune only specific regions of the image. Although local optimization in general wins out, image navigation can only be shown to be more effective in the SANMIGUEL20 image.

On the other hand, the behavior of participants during the trials and the responses in the questionnaire suggests that participants much prefer the image navigation interface and dread working with the sliders. Participants rated the pleasantness of using image navigation with a 4.5 on a scale of 1 to 5 where 5 means most pleasant. The slider interface on the other hand received an average rating of 2.7. It could be argued that participants may be more familiar and proficient with slider-based interfaces, but in the questionnaire they indicated that both interfaces were equally easy to learn. To explain this inconsistency we reevaluated the properties of the noise in the scenes and what type of filters participants had applied to filter this noise, and we realized that the in-camera depth of field or motion blur noise in both the POOLBALL and SHIP images could be very easily reduced even with very basic filters like Gaussian blur. Users would simply ignore all the complex filters with many sliders in favor of this particular filter.

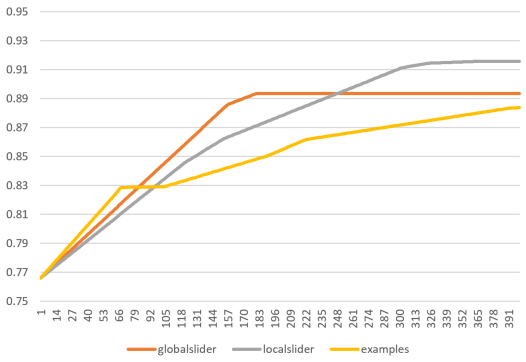
To quantify this behaviour we also tracked changes to the SSIM score while a user was working on improving an image. By aggregating the SSIM trend per method across all sessions, we can visualize how users work on different types of images. The trends in the SANMIGUEL20 and SHIP scenes are visualized in Figure 11.1. The SANMIGUEL20 scene requires the complex cross-bilateral filter to achieve the highest quality results and the example hierarchy approach lets users tune it much faster than the slider based approaches. On the other hand, the SHIP scene can be easily improved with just the Gaussian blur filter for the depth-of-field effect and sliders prove to be more effective to quickly optimize such a simple filter.

Table 11.1: Mean SSIM of final filtered images in initial case study. The results are aggregated by scene and workflow.

	Workflow	**Final SSIM**
POOLBALL	No filters	0.880
	Sliders (global)	0.673
	Sliders (local)	0.939
	Image navigation	0.864
	Optimal choice	0.997
SANMIGUEL20	No filters	0.422
	Sliders (global)	0.665
	Sliders (local)	0.726
	Image navigation	0.729
	Optimal choice	0.984
SHIP	No filters	0.755
	Sliders (global)	0.893
	Sliders (local)	0.915
	Image navigation	0.883
	Optimal choice	0.998



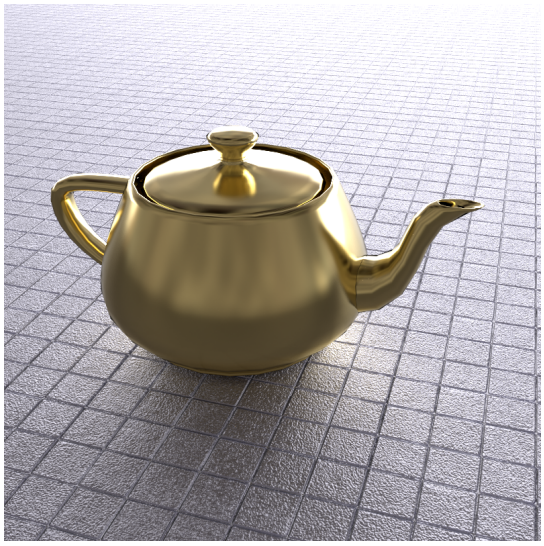
(a) SANMIGUEL20 scene requires complex filters and users tune these faster with the example hierarchy.



(b) SHIP scene can easily be improved with just Gaussian blur and such a simple filter is easily tuned with sliders.

Figure 11.1: Comparison of SSIM through time for the SANMIGUEL20 and SHIP scenes averaged across all sessions per method.

(a) Noisy render



(b) Ground truth image

Figure 11.2: TEAPOT scene

11.2. Case Study Extension

The observation that simple filters were already sufficient for significantly reducing the noise in the POOLBALL and SHIP images prompted us to repeat the case study with a different set of images. We selected new images from a set of candidates by finding the most *complex* ones. Complexity was evaluated by writing a small program that generates a filter bank and evaluates how error reduction is achieved by only applying the best three filters. A better error reduction would mean that less filters are necessary to clean up the image, so these images were assigned a low complexity rating. The top three of these candidates are the following images:

- TEAPOT: A scene with a very detailed floor texture that rapidly changes as it fades into the distance. A teapot is positioned on the floor that has a very different smooth surface. (Figure 11.2)
- SANMIGUEL25: The same scene as SANMIGUEL20, but with a different viewpoint that contains a wide variety of geometric details rather than an abundance of leaves. (Figure 11.3)
- DRAGONFOG: A scene that combines a depth-of-field effect with highly detailed geometry and participating media lighting streaks. (Figure 11.4)

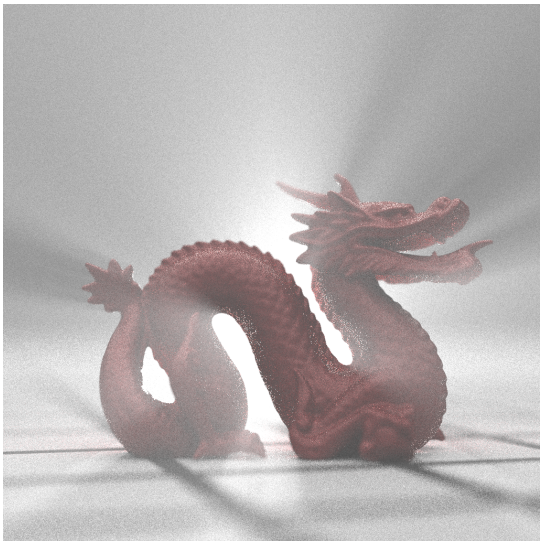
We expect that users will not be able to get away with application of simple filters in this case and that if they choose to do so with the slider-interface, the image navigation that hides these sliders will result in better filter choices. The user study was repeated with the same tutorial and trial procedures, but with the new set of three images and without the questionnaire. This time six users participated in the study, resulting in two samples per pair of image and method. Two of the participants had also participated in the previous case study, but we do not believe that this made a difference given that one month had passed since. The quantitative results of this extension are listed in Table 11.2.

Our hypothesis that participants would now start using more complex filters was confirmed. Observation indicated that they spent significantly more time on tuning parameters of the more advanced filters for these new images. Local optimization through image navigation is now superior in two out of three images. The SANMIGUEL25 image could only be effectively filtered with the cross-bilateral filter, which has a large number of sliders and represents a perfect scenario for the image navigation approach. Global slider optimization is quantitatively the most effective approach in the remaining DRAGONFOG. It turned out that our image selection criteria had failed, because this image was a near-perfect case for the guided filter. The depth of field effect in the foreground and background coincided with constant normals and the in-focus dragon had rapidly varying normals, resulting in a perfect noise variance metric for the guided filter. The performance of image navigation in this scene was even more surprising, but this was the cause of outlier participants who ran out of time while optimizing only a small part of the image. With those outliers removed, the example

(a) Noisy render

(b) Ground truth image

Figure 11.3: SANMIGUEL25 scene



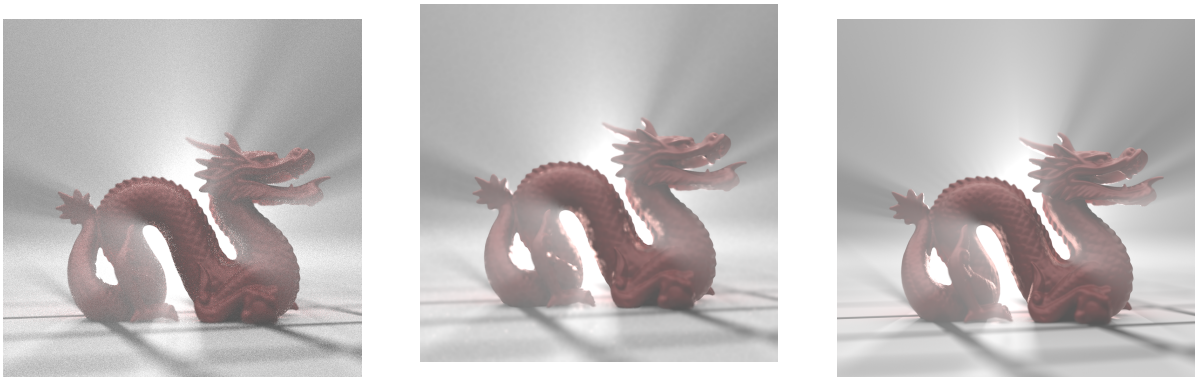
(a) Noisy render

(b) Ground truth image

Figure 11.4: DRAGON scene

Table 11.2: Mean SSIM of final filtered images in case study extension. The results are aggregated by scene and workflow.

	Workflow	Final SSIM
DRAGONFOG	No filters	0.434
	Sliders (global)	0.970
	Sliders (local)	0.914
	Image navigation	0.688
	Image navigation (no outliers)	0.963
	Optimal choice	0.998
SANMIGUEL25	No filters	0.145
	Sliders (global)	0.354
	Sliders (local)	0.340
	Image navigation	0.360
	Optimal choice	0.909
TEAPOT	No filters	0.404
	Sliders (global)	0.609
	Sliders (local)	0.667
	Image navigation	0.671
	Optimal choice	0.983



(a) Noisy render

(b) Best filtered image (using global slider)

(c) Ground truth

Figure 11.5: Comparison of noisy input image, best filtered image and ground truth for DRAGON scene in case study extension.

hierarchy is seen to be performing better than the local sliders, but not as good as the global slider for the aforementioned reason.

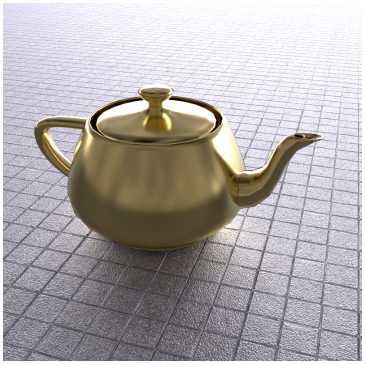
11.3. Efficacy

Although the performance of image navigation compared to slider-based interfaces remains somewhat inconclusive, users are content with the filtered results that they can achieve within the short time span of 5 minutes. We show a comparison between the input, ground truth and best filtered result across all participants for every image in the extended case study in figures 11.5, 11.6 and 11.7. This comparison shows that users can effectively select and apply appropriate filters to significantly improve the rendered images.

In the case study we limited participants to a time span of 5 minutes, which yields a lower bound for the performance. To also estimate the upper bound, we determined the result in the case where the perfect filter is chosen for every pixel, as seen in Figure 11.8. Although it demonstrates that even just 8 filters are sufficient for high quality results, the filter choice map is still much too incoherent to be painted by a human. However, Bauszat et al. [7] have shown that optimizing a filter choice map for coherency only has a minor effect on the quality of the final result so we believe that similarly high quality results are achievable when users take more time to more precisely paint filters.

(a) Noisy render

(b) Best filtered image (using local slider)



(c) Ground truth

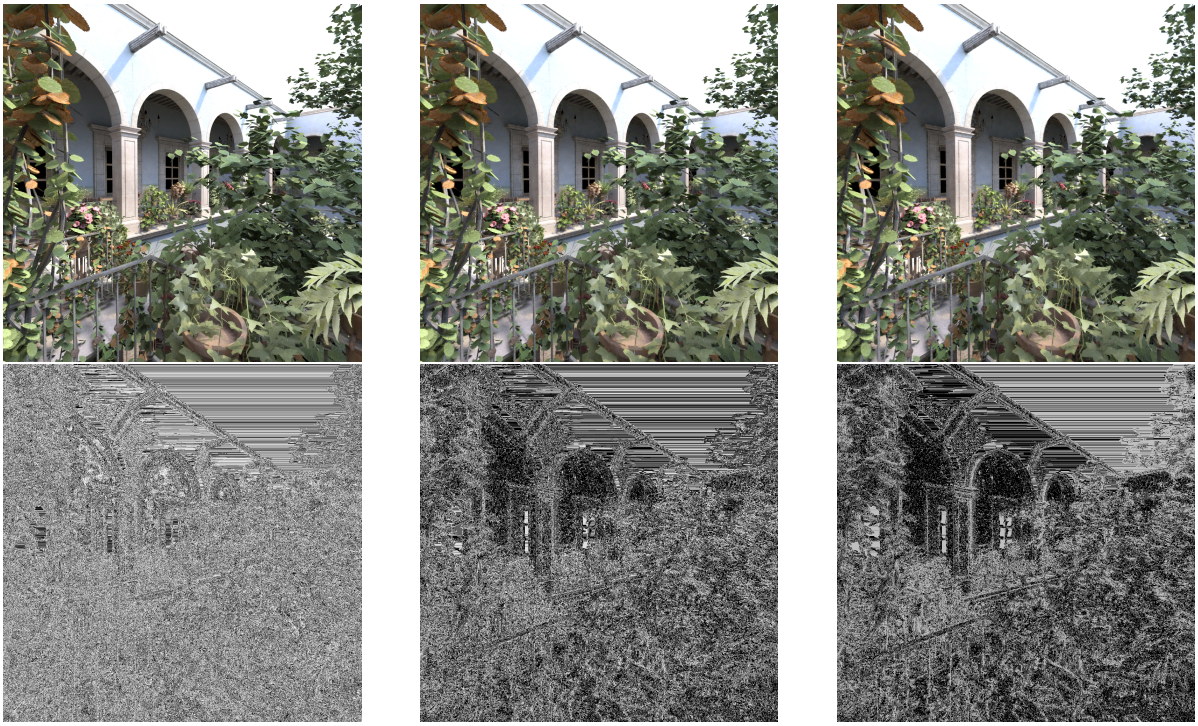
Figure 11.6: Comparison of noisy input image, best filtered image and ground truth for TEAPOT scene in case study extension.

(a) Noisy render

(b) Best filtered image (using image navigation)

(c) Ground truth

Figure 11.7: Comparison of noisy input image, best filtered image and ground truth for SANMIGUEL25 scene in case study extension.



(a) Optimal filter choice when using all 64 filters (SSIM = 0.972).

(b) Optimal filter choice using only 8 best filters (SSIM = 0.927).

(c) Optimal filter choice using only 4 best filters (SSIM = 0.892).

Figure 11.8: Comparison of results and filter choice maps when allowing 64, 8 or 4 filters to be used to filter the SANMIGUEL20 scene.

11.4. Limitations

The limited number of samples in our case study has resulted in some problems with statistical significance, primarily exemplified by the outlier for image navigation in the DRAGONFOG image. We have therefore opted not to quantify the statistical significance of this study. Another problem with comparison of the slider and image navigation based interfaces was that most of the filters that we were able to implement still had a low number of parameters (≤ 2). Only the cross-bilateral filter had a large parameter space and this filter was simply avoided by many participants, however we believe that this has given image navigation a competitive edge in the SANMIGUEL20 and SANMIGUEL25 images where the cross-bilateral filter is the best performing filter.

During our case study we mimicked a consistent mental ground truth by providing all participants with the actual ground truth image. We do not believe that this introduced a bias, because actual 3D artists have in-depth knowledge of the scenes that they are working with and know what the various materials and lighting conditions should look like. It is interesting to note that some participants indicated that they were actually hindered by the ground truth goal, because they preferred a different look. This most notably occurred with the POOLBALL image where some participants preferred the pool ball surface to be overblurred because they thought the high frequency texture looked unrealistic. This observation supports the notion that a mental ground truth as artistic vision may actually be preferable to the actual ground truth in real-life scenarios.

Some users suggested that they were missing features to compare different filter choices more easily. They would have liked to have a palette of previously chosen filters to quickly evaluate possible new choices against. This may have caused users to sometimes downgrade the image by selecting new filters that were actually less optimal.

We would like to restate that the benefit of our black-box approach to filters is that we can also include state-of-the-art error estimation frameworks into our filter bank for a theoretical guarantee that the results of our workflow are at least as close to the ground truth as these. On the flip side, the results of our framework are still constrained by the best filters in the filter bank, which may be lackluster due to inherent filter limits or an unfortunate sampling of parameters (see Figure 11.7 for an example). Through experimentation we found

that perfectly tuning even the basic bilateral filter can lead to results that are almost indistinguishable from the ground truth, so we argue that a low quality filter bank merely requires the user to paint filters with smaller granularity rather than making it impossible to achieve results that are close to the ground truth. Whether the required granularity is feasible in practice remains another matter. For example, Figure 11.8 shows that a high variance filter choice map may be required in some scenarios for best possible results.

12

Future Work

Time constraints prevented us from pursuing various research avenues. We also believe that our combination of image navigation and local optimization can be applied to a wide variety of other problems. We describe both of these types of possible further research in this section.

12.1. Case Study

Repeating our case study with a larger and more diverse group of participants and using more complex filters would likely lead to more conclusive results regarding the effectiveness of image navigation. This new study should involve more filters with high dimensional parameter spaces like BM3D.

12.2. Filter Bank Generation

One of the major shortcomings in our work remains the generation of appropriate filter examples. Increased variance and increased coverage of the filtered image space are necessary to make user's choices easier. Our LHS approach treats a parameter space as a linear mapping to output images, whereas many filter parameters control edge functions where small changes can result in rapidly changing results. We believe that our workflow would benefit from adaptively optimizing sample allocation [18].

12.3. Meta-filters

An interesting consequence of our black-box approach to filter usage is that it would also be possible to include existing filter tuning frameworks into our filter bank, where the user navigates the tuning parameter space rather than the parameter space of the filters directly. By including these *meta-filters* and ensuring that users are able to successfully navigate to these, we could guarantee that the results of our workflow are always at least as close to the ground truth as the state-of-the-art, and the user can spend their time on correcting only the remaining flaws in the results of these frameworks.

12.4. Visualization Aids

Despite the visualization efforts, users still found it challenging to determine if a given filter really makes a difference compared to previously considered choices. Case study participants suggested having a palette of previously chosen filters that users could compare to or fall back to when trying to find a new filter.

12.5. Temporal Extrapolation

To make our workflow economically viable for the digital animation industry, we will also have to extend it to be more scalable with regards to human effort. Therefore we believe that a learning system should be integrated [28]. Such a system could learn from one or more frames in a shot and learn from this to extrapolate the filter choices across all the frames in the same shot or even across similar shots. At the minimum, a form of patch matching like the non-local means filter, could be used to choose the most likely intended filter. A film like Toy Story 3, for example, has approximately 80 frames per shot [1] on average (lowest measured seconds per shot multiplied by frame rate). Only having to paint one or two of these would result in a significant reduction of work.

12.6. Application to Real-Time Rendering

In this research we have only touched upon offline rendering, but it is evident that fast filters like the guided filter can already be used for high quality real-time Monte Carlo rendering [6]. Modern game development workflows already include tuning of post-processing effects like real-time color correction depending on scene and gameplay state [38]. We believe that our filter tuning software can be integrated in a similar fashion, where artists can select the optimal filter(s) for scenes in a game using image navigation.

12.7. Plugins for Existing Software

Artists are already familiar with existing tools that are used for post-processing like Adobe Photoshop [3], Adobe After Effects [2] and The Foundry's Nuke [19]. We believe that it may be beneficial to integrate our workflow into this existing software so that they can take advantage of industry standard tools for operations like selection and brushing.

12.8. Image Navigation for Other Problems

The original design galleries paper already shows how easily the image-navigation approach can be extended to other problems than image processing. For example, we believe that our workflow could also be applied to geometry modeling problems like mesh simplification, where users choose simplification parameters by image previews and then select subsets of a model to locally optimize with different parameters [31].

13

Conclusion

We have presented a new interface for filtering Monte Carlo rendered images by hand. The interface allows users to paint filter parameters directly to an image for more flexibility and allows for novel filter blending operations. We have introduced an interactive workflow with task-focused stages to incorporate this to reduce the cognitive workload of the user.

We have shown through a case study how image navigation allows novice users to navigate the high dimensional parameter space of filters. Users can produce higher quality results with our local parameter painting approach compared to global filter tuning. Our image navigation paradigm allows users to better approximate the ground truth for images that require filters with a high dimensional parameter space, but we were unable to show that image navigation is superior to sliders in general. We expect that future research with a larger scale user study and more complex filters will provide a more conclusive outcome.

Bibliography

[1] Cinemetrics database. http://www.cinemetrics.lv/database.php. Accessed: 2017-09-06.

[2] Adobe. Adobe after effects. http://www.adobe.com/products/aftereffects.html, . Accessed: 2017-09-06.

[3] Adobe. Adobe photoshop. http://www.adobe.com/products/photoshop.html, . Accessed: 2017-09-06.

[4] James Arvo and David Kirk. A survey of ray tracing acceleration techniques. *An introduction to ray tracing*, pages 201–262, 1989.

[5] Adam Baker. Learning from photoshop's "variations" tool. http://www.merges.net/theory/20010308.html. Accessed: 2017-09-11.

[6] Pablo Bauszat, Martin Eisemann, and Marcus Magnor. Guided image filtering for interactive high-quality global illumination. In *Computer Graphics Forum*, volume 30, pages 1361–1368. Wiley Online Library, 2011.

[7] Pablo Bauszat, Martin Eisemann, Elmar Eisemann, and Marcus Magnor. General and robust error estimation and reconstruction for monte carlo rendering. In *Computer Graphics Forum*, volume 34, pages 597–608. Wiley Online Library, 2015.

[8] Pamela Baxter and Susan Jack. Qualitative case study methodology: Study design and implementation for novice researchers. *The qualitative report*, 13(4):544–559, 2008.

[9] Gary Bradski. The opencv library. *Dr. Dobb's Journal: Software Tools for the Professional Programmer*, 25 (11):120–123, 2000.

[10] Antoni Buades, Bartomeu Coll, and J-M Morel. A non-local algorithm for image denoising. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 2, pages 60–65. IEEE, 2005.

[11] Brent Burley and Walt Disney Animation Studios. Physically-based shading at disney. In *ACM SIGGRAPH*, volume 2012, pages 1–7, 2012.

[12] Omar Cornut. imgui library. https://github.com/ocornut/imgui. Accessed: 2017-09-14.

[13] Cyril Crassin, Fabrice Neyret, Miguel Sainz, Simon Green, and Elmar Eisemann. Interactive indirect illumination using voxel cone tracing. In *Computer Graphics Forum*, volume 30, pages 1921–1930. Wiley Online Library, 2011.

[14] Kostadin Dabov, Alessandro Foi, Vladimir Katkovnik, Karen Egiazarian, et al. Image denoising with block-matching and 3 d filtering. In *Proceedings of SPIE*, volume 6064, pages 606414–606414, 2006.

[15] Mauricio Delbracio, Pablo Musé, Antoni Buades, Julien Chauvier, Nicholas Phelps, and Jean-Michel Morel. Boosting monte carlo rendering by ray histogram fusion. *ACM Trans. Graph.*, 33(1):8–1, 2014.

[16] David L Donoho and Jain M Johnstone. Ideal spatial adaptation by wavelet shrinkage. *biometrika*, 81(3): 425–455, 1994.

[17] Elmar Eisemann and Frédo Durand. Flash photography enhancement via intrinsic relighting. *ACM transactions on graphics (TOG)*, 23(3):673–678, 2004.

[18] Pierre Etoré and Benjamin Jourdain. Adaptive optimal allocation in stratified sampling methods. *Methodology and Computing in Applied Probability*, 12(3):335–360, 2010.

[19] The Foundry. The foundry's nuke. https://www.foundry.com/products/nuke. Accessed: 2017-09-06.

[20] GitHub. Rendering and diffing images. https://help.github.com/articles/rendering-and-diffing-images/. Accessed: 2017-09-14.

[21] Gizmodo. One animal in zootopia has more individual hairs than every character in frozen combined. http://io9.gizmodo.com/one-animal-in-zootopia-has-more-individual-hairs-than-e-1761542252. Accessed: 2017-09-10.

[22] Kate Gregory and Ade Miller. C++ amp: accelerated massive parallelism with microsoft visual c++. 2014.

[23] Kaiming He, Jian Sun, and Xiaoou Tang. Guided image filtering. *IEEE transactions on pattern analysis and machine intelligence*, 35(6):1397–1409, 2013.

[24] Daniel Heesch. A survey of browsing models for content based image retrieval. *Multimedia Tools and Applications*, 40(2):261–284, 2008.

[25] Stefan Jeschke, David Cline, and Peter Wonka. A gpu laplacian solver for diffusion curves and poisson image editing. In *ACM Transactions on Graphics (TOG)*, volume 28, page 116. ACM, 2009.

[26] James T Kajiya. The rendering equation. In *ACM Siggraph Computer Graphics*, volume 20, pages 143–150. ACM, 1986.

[27] Nima Khademi Kalantari and Pradeep Sen. Removing the noise in monte carlo rendering with general image denoising algorithms. In *Computer Graphics Forum*, volume 32, pages 93–102. Wiley Online Library, 2013.

[28] Nima Khademi Kalantari, Steve Bako, and Pradeep Sen. A machine learning approach for filtering monte carlo noise. *ACM Trans. Graph.*, 34(4):122–1, 2015.

[29] Alexander Keller, Luca Fascione, Marcos Fajardo, Iliyan Georgiev, Per H Christensen, Johannes Hanika, Christian Eisenacher, and Gregory Nichols. The path tracing revolution in the movie industry. In *SIGGRAPH Courses*, pages 24–1, 2015.

[30] William B Kerr and Fabio Pellacini. Toward evaluating material design interface paradigms for novice users. In *ACM Transactions on Graphics (TOG)*, volume 29, page 35. ACM, 2010.

[31] Youngihn Kho and Michael Garland. User-guided simplification. In *Proceedings of the 2003 symposium on Interactive 3D graphics*, pages 123–126. ACM, 2003.

[32] Santhana Krishnamachari and Mohamed Abdel-Mottaleb. Image browsing using hierarchical clustering. In *Computers and Communications, 1999. Proceedings. IEEE International Symposium on*, pages 301–307. IEEE, 1999.

[33] Eric P Lafortune and Yves D Willems. Bi-directional path tracing. 1993.

[34] Tzu-Mao Li, Yu-Ting Wu, and Yung-Yu Chuang. Sure-based optimization for adaptive sampling and reconstruction. *ACM Transactions on Graphics (TOG)*, 31(6):194, 2012.

[35] Joe Marks, Brad Andalman, Paul A Beardsley, William Freeman, Sarah Gibson, Jessica Hodgins, Thomas Kang, Brian Mirtich, Hanspeter Pfister, Wheeler Ruml, et al. Design galleries: A general approach to setting parameters for computer graphics and animation. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 389–400. ACM Press/Addison-Wesley Publishing Co., 1997.

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

[37] Michael D McKay, Richard J Beckman, and William J Conover. Comparison of three methods for selecting values of input variables in the analysis of output from a computer code. *Technometrics*, 21(2): 239–245, 1979.

[38] Jason Mitchell, Gary McTaggart, and Chris Green. Shading in valve's source engine. In *Acm siggraph 2006 courses*, pages 129–142. ACM, 2006.

[39] Bochang Moon, Nathan Carr, and Sung-Eui Yoon. Adaptive rendering based on weighted local regression. *ACM Transactions on Graphics (TOG)*, 33(5):170, 2014.

[40] Kenneth Moreland. Diverging color maps for scientific visualization. *Advances in Visual Computing*, pages 92–103, 2009.

[41] Addy Ngan, Frédo Durand, and Wojciech Matusik. Image-driven navigation of analytical brdf models. In *Rendering Techniques*, pages 399–407, 2006.

[42] Fred E Nicodemus. Directional reflectance and emissivity of an opaque surface. *Applied optics*, 4(7): 767–775, 1965.

[43] Ryan S Overbeck, Craig Donner, and Ravi Ramamoorthi. Adaptive wavelet rendering. *ACM Trans. Graph.*, 28(5):140–1, 2009.

[44] Kirsty Parkin. Building 3d with ikea. *Cgsociety. org. Retrieved December*, 10:2014, 2014.

[45] Matt Pharr, Wenzel Jakob, and Greg Humphreys. *Physically based rendering: From theory to implementation*. Morgan Kaufmann, 2016.

[46] Pixar. Denoising - renderman user guide. `https://renderman.pixar.com/resources/RenderMan_20/risDenoise.html`. Accessed: 2017-09-11.

[47] Stephen M Pizer, E Philip Amburn, John D Austin, Robert Cromartie, Ari Geselowitz, Trey Greer, Bart ter Haar Romeny, John B Zimmerman, and Karel Zuiderveld. Adaptive histogram equalization and its variations. *Computer vision, graphics, and image processing*, 39(3):355–368, 1987.

[48] Riccardo Poli and Stefano Cagnoni. Genetic programming with user-driven selection: Experiments on the evolution of algorithms for image enhancement. *Genetic Programming*, pages 269–277, 1997.

[49] Fabrice Rousselle, Claude Knaus, and Matthias Zwicker. Adaptive sampling and reconstruction using greedy error minimization. In *ACM Transactions on Graphics (TOG)*, volume 30, page 159. ACM, 2011.

[50] Carl-Erik Särndal, Bengt Swensson, and Jan Wretman. *Model assisted survey sampling*. Springer Science & Business Media, 2003.

[51] Benjamin Scheibehenne, Rainer Greifeneder, and Peter M Todd. Can there ever be too many options? a meta-analytic review of choice overload. *Journal of Consumer Research*, 37(3):409–425, 2010.

[52] Christoph Schied, Anton Kaplanyan, Chris Wyman, Anjul Patney, Chakravarty R Alla Chaitanya, John Burgess, Shiqiu Liu, Carsten Dachsbacher, Aaron Lefohn, and Marco Salvi. Spatiotemporal variance-guided filtering: real-time reconstruction for path-traced global illumination. In *Proceedings of High Performance Graphics*, page 2. ACM, 2017.

[53] Charles M Stein. Estimation of the mean of a multivariate normal distribution. *The annals of Statistics*, pages 1135–1151, 1981.

[54] Carlo Tomasi and Roberto Manduchi. Bilateral filtering for gray and color images. In *Computer Vision, 1998. Sixth International Conference on*, pages 839–846. IEEE, 1998.

[55] Thomas Torsney-Weir, Ahmed Saad, Torsten Moller, Hans-Christian Hege, Britta Weber, Jean-Marc Verbavatz, and Steven Bergner. Tuner: Principled parameter finding for image segmentation algorithms using visual response surface exploration. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):1892–1901, 2011.

[56] Dimitri Van De Ville and Michel Kocher. Sure-based non-local means. *IEEE Signal Processing Letters*, 16 (11):973–976, 2009.

[57] Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing*, 13(4):600–612, 2004.

[58] Joss Whittle, Mark W Jones, and Rafał Mantiuk. Analysis of reported error in monte carlo rendered images. *The Visual Computer*, pages 1–9, 2017.

[59] Matthias Zwicker, Wojciech Jarosz, Jaakko Lehtinen, Bochang Moon, Ravi Ramamoorthi, Fabrice Rousselle, Pradeep Sen, Cyril Soler, and S-E Yoon. Recent advances in adaptive sampling and reconstruction for monte carlo rendering. In *Computer Graphics Forum*, volume 34, pages 667–681. Wiley Online Library, 2015.

[60] Atılım Çetin. Guided filter library. `https://github.com/atilimcetin/guided-filter`. Accessed: 2017-09-14.