Discovering New Monte Carlo Noise Filters with Genetic Programming

Peter Kán , Maxim Davletaliyev and Hannes Kaufmann

Institute of Software Technology and Interactive Systems, Vienna University of Technology, Vienna, Austria

Abstract

This paper presents a novel method for the discovery of new analytical filters suitable for filtering of noise in Monte Carlo rendering. Our method utilizes genetic programming to evolve the set of analytical filtering expressions with the goal to minimize image error in training scenes. We show that genetic programming is capable of learning new filtering expressions with quality comparable to state of the art noise filters in Monte Carlo rendering. Additionally, the analytical nature of the resulting expressions enables the run-times one order of magnitude faster than compared state of the art methods. Finally, we present a new analytical filter discovered by our method which is suitable for filtering of Monte Carlo noise in diffuse scenes.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: 3D Graphics and Realism—Raytracing

1. Introduction

Photorealistic image synthesis has been one of the main challenges in computer graphics research since its beginning. Monte Carlo rendering seems to be the most suitable solution due to its physical plausibility, impressive level of photorealism, and versatility for numerous visual effects. The main drawback of Monte Carlo rendering is its high computational cost due to the vast amount of samples required to compute the multidimensional light integral. Moreover, Monte Carlo rendering utilizes stochastic sampling which leads to high variance for low sample count. This high variance appears as noise in the final image. Previous research demonstrated that multidimensional filtering is a viable solution for removing noise from Monte Carlo rendering [SD11, RKZ12, KBS15, ZJL*15]. However, most of these methods still require computational time in terms of seconds, thus prohibiting real-time performance.

In order to address this problem, we suggest to search for analytical expressions for high-quality filtering which can be rapidly executed on the GPU. Our search is based on evolutionary computation. We use genetic programming [Koz92] to optimize the set of filtering expressions with the goal to minimize the image error with respect to ground truth. Analytical expressions were also used in early work on noise filtering of color images. However, noisy color data from Monte Carlo rendering is insufficient to filter the noise which was formed by stochastic sampling of the high-dimensional integral. We overcome this problem, similarly to previous methods [SD11, KBS15, ZJL*15], by utilizing additional scene features (e.g. world positions, normals, direct illumination, etc.). Moreover, we also employ gradients and variances of features.

Our genetic programming framework identifies new filters in an iterative optimization process. Firstly, the set of filtering expressions (population) is initialized by known analytical filters. The following steps are then performed in each iteration: The relative mean square error (RelMSE) [RKZ11] is calculated for each expression on the set of training scenes. The expressions are then altered by two expansion operations: (1) Crossover, which generates a new expression from two selected parents and (2) mutation, which randomly perturbs individual expressions based on a specified grammar (Figure 2). Input filters for these expansion operations are selected from population by tournament selection (Section 4.4). This process generates filters optimized for Monte Carlo noise filtering. Numerous scenes are used in the training process to generate filters applicable for general Monte Carlo rendering of arbitrary scene.

We demonstrate the applicability of genetic programming for discovery of analytical filters by presenting a generated filtering expression suitable for filtering of Monte Carlo noise in diffuse

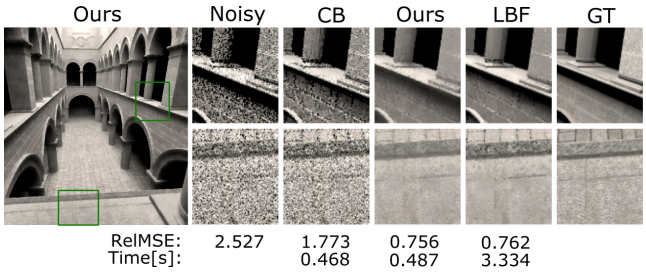

Figure 1: *Filtering results of noisy rendering with 4 samples per pixel. The comparison shows the cross-bilateral filter (CB), our filter, learning-based filtering (LBF) [KBS15] and ground truth (GT).*

scenes. The presented filtering expression is evaluated in terms of quality and speed, and compared to state of the art [KBS15]. Our results show that the discovered expression can achieve comparable quality to state of the art while running one order of magnitude faster. Finally, due to the analytical nature of the presented filter, it can be easily parallelized and implemented on the GPU. The main contributions of this paper can be summarized as following:

- We present a novel method for discovery of new analytical Monte Carlo noise filters.
- A new efficient analytical filter, resulting from our genetic programming experiments, is demonstrated.
- We identify the set of most important scene features for Monte Carlo noise filtering.

2. Related Work

Previous research demonstrated numerous methods for multidimensional noise filtering in Monte Carlo rendering. A multidimensional adaptive sampling and reconstruction method was proposed by Hachisuka et al. [HJW*08]. This method leverages multiple dimensions of sampling space to distinguish between noise caused by variance and discontinuities in geometry and textures. A disadvantage of this algorithm is its inefficiency with high dimensionality. Lehtinen et al. presented methods for reconstruction of spatial [LALD12] and temporal [LAC*11] light field from which the final image is reconstructed. Rousselle et al. proposed two algorithms for adaptive sampling and reconstruction [RKZ11, RKZ12] which increase the quality of predictive rendering.

The capabilities of a wavelet basis in adaptive multidimensional Monte Carlo rendering were investigated by Overbeck et al. [ODR09]. Previous research also showed that local regression theory can be utilized for adaptive sampling and filtering in Monte Carlo rendering [MCY14, BRM*16] to correctly estimate the importance of specific features for local filtering. Sen and Darabi [SD11] proposed a method which uses statistical information to recognize the noise caused by random parameters. This information can be then used in a cross-bilateral filter to preserve important scene details. Additionally, a specific filter for motion blur rendering was presented in [ETH*09]. Recently, Kalantari et al. [KBS15] investigated the capabilities of neural networks for Monte Carlo noise filtering. The authors trained a neural network to derive filter parameters from scene features and adaptively filter the image in high quality. In our evaluation, we compare our filtering expression to their work.

An axis-aligned filtering algorithm for interactive physically-based global illumination was proposed in [MWRD13]. This method focuses only on the diffuse indirect illumination. An adaptive real-time filtering algorithm was proposed by Gastal and Oliveira [GO12]. The authors compute filter responses at the reduced set of sampling points and use them for interpolation. Recently, a real-time filtering method for distributed effects was presented by Yan et al. [YMRD15]. The authors use four 1D filters to approximate 4D sheared filter. The limitation of this method is that it cannot filter the noise from multiple distributed effects simultaneously. A comprehensive overview of methods for adaptive sampling and reconstruction in Monte Carlo rendering can be found in [ZJL*15].

$$
\begin{aligned}
\langle expression \rangle &::= \langle node \rangle \\
\langle node \rangle &::= \langle op \rangle \mid \langle scalar \rangle \\
\langle op \rangle &::= \langle unaryOp \rangle (\langle node \rangle) \\
&\mid \langle binaryOp \rangle (\langle node \rangle, \langle node \rangle) \\
&\mid \langle vectorOp \rangle (\langle vector \rangle, \langle vector \rangle) \\
\langle unaryOp \rangle &::= -\mid sin \mid cos \mid tan \mid exp \mid asin \mid acos \mid atan \mid sqrt \\
&\mid mitchell \mid sinc \mid epanechnikov \mid biweight \mid tricube \\
\langle binaryOp \rangle &::= *\mid -\mid +\mid /\mid pow \\
\langle vectorOp \rangle &::= dot \mid distance2 \mid distance1 \mid distanceMax \\
\langle vector \rangle &::= worldPosition \mid normal \mid texture \mid secTexture \\
&\mid depth \mid dirIllumination \mid wpGradient \mid nGradient \\
&\mid tGradient \mid secTGradient \mid dGradient \mid diGradient \\
\langle scalar \rangle &::= \langle variable \rangle \mid \langle const \rangle \\
\langle variable \rangle &::= wpVariance \mid nVariance \mid texVariance \\
&\mid secTexVariance \mid dVariance \mid diVariance \\
\langle const \rangle &::= 0.1 \mid 0.2 \mid 0.3 \mid 0.3333 \mid 1.0 \mid 2.0 \mid 3.0 \mid \pi
\end{aligned}
$$

Figure 2: *The grammar used for generation of new expressions.*

Genetic programming [Koz92] is a class of adaptive stochastic optimization algorithms involving search and optimization in the space of possible programs or analytic expressions. Genetic programming was previously used in computer graphics for generating new BRDF expressions [BLPW14] or for optimizing ray-triangle intersection calculation [KS06].

3. Analytical Filtering Expressions

General image-space filtering algorithms estimate the value of pixel i as a weighted average of its neighborhood:

$$\hat{c}_i = \frac{\sum_{j \in \mathcal{N}_i} c_j w(i,j)}{\sum_{j \in \mathcal{N}_i} w(i,j)} \tag{1}$$

where $\hat{c}_i$ is the filtered color value, c_j is the noisy color input and $\mathcal{N}_i$ is the neighborhood centered on position i. $w(i,j)$ is the weight of contribution of pixel j to the estimate. The calculation of $w(i,j)$ is the crucial part of image filtering algorithms. Our method calculates this weight by analytical filtering expressions which are to be optimized in our genetic programming search.

During optimization, each of these expressions is represented in the form of an Abstract Syntax Tree (AST) which allows to perform crossover, mutations, and execution of expressions. The mathematical operators of the expressions are represented as inner nodes of the AST and the feature data and constants are leaf nodes. Then, any equation can be represented as a tree in which the parameters of a function are its child nodes. In order to generate only valid expressions which contain operators often used in filters, we designed a specific grammar (Figure 2). Each expression in our search conforms to this grammar.

4. Genetic Programming Search

Our genetic programming framework trains the set of new filtering expressions in an iterative optimization process. Firstly, the set of filtering expressions is initialized by the bilateral filter and its random mutations (Section 4.2). Then, the filters are iteratively optimized with the goal to minimize the error of the filtered image. Each iteration of our optimization consists of the following steps: (1) Evaluation of the fitness function, (2) selection of filters which will proceed to the next iteration, (3) generation of new expressions by crossover, and (4) altering the expressions by mutations.

We employ an island model genetic algorithm which favors exploration of the space of possible filters over narrowly searching within profitable regions. The island model genetic algorithm subdivides the whole population of expressions to multiple subpopulations (islands). All islands are evolving separately with rare interactions between them. These interactions typically include migration of the best expressions amongst islands. As a migration strategy, we use an approach similar to Brady et al. [BLPW14] which migrates the best expression from each island to the next island in each n-th iteration. We set n to 5 in our implementation. We used four islands in our experiments each containing 100 filters.

4.1. Crossover

The crossover operation is used to generate new filters from two randomly selected parents. A new filter is generated from two parent expressions in three steps: For each parent expression a node in its AST tree is randomly selected (a crossover point). Then, the sub-trees below the selected crossover points are swapped between parents. Finally, one of these new expressions is used as a result.

4.2. Mutations

In order to further explore the space of possible solutions, genetic programming uses mutations to alter individual expressions. One mutation is randomly selected and applied to a filtering expression. We use the following mutations in our framework:

- *replace* - replaces random node from AST representation of an expression by a compatible node from our grammar (Figure 2). The child nodes of the replaced node stay untouched and become the children of the new node.
- *swap* - randomly selects two compatible nodes within AST and swaps their sub-trees. The swap mutation is performed only if one node is not the sub-tree of the other one.
- *insert* - randomly selects one node from AST and substitutes it with an expression from codebook. In this case, the whole subtree below the selected node is substituted by a new expression.
- *delete* - randomly selects a node from AST and replaces it by a constant node with value 1.

Codebook. Many efficient analytical noise filters were presented in previous research. We can leverage these filters by enabling our algorithm to insert them or their sub-parts into the optimized expressions. Therefore, the *insert* mutation utilizes a codebook of analytical expressions to augment the filters during optimization. The codebook in our method is formed by using the following analytical filters and all of their mathematical sub-expressions: Mitchell, sinc, Epanechnikov, biweight, tricube, Gaussian, and cross-bilateral.

4.3. Fitness Function

The fitness function is the most important part of our method which learns new filtering expressions by minimizing this function. The main goal of Monte Carlo noise filtering is to produce the filtered image as close as possible to the ground truth solution (calculated with high sampling rate without filtering). Therefore, our fitness function measures the error between the image filtered by a specific filtering expression and the ground truth. In our experiments we used the relative mean square error (RelMSE) [RKZ11] which is commonly used in Monte Carlo rendering. RelMSE is calculated for each color of a pixel as $(img - ref)^2/(ref^2 + \varepsilon)$ where img and ref are filtered and ground truth values of the pixel and $\varepsilon = 0.01$ is used to prevent over-weighting of errors in dark regions. We used 6 training scenes in our experiments and the fitness function for each expression was calculated as the sum of errors in all these scenes.

Training scenes. In order to learn general filtering expressions independent of the specific scene, it is desirable to use many training scenes in the optimization process. We used six scenes in our experiments. Three of these scenes were rendered by the pbrt renderer and the other three by NVIDIA OptiX. For each scene we rendered a noisy input image with 4 samples per pixel and the ground truth image with high sampling rate. Additionally, we rendered the following scene features with 4 samples per pixel: World position, normal, diffuse texture, secondary diffuse texture (i.e. diffuse texture visible at glossy reflection), depth, direct illumination, gradients of all mentioned features, and variances of all features. This data was then used in our genetic programming optimization as the terms of input grammar during evaluation of filtering expressions (Figure 2). Training scenes are shown in supplementary material.

4.4. Selection

A new generation of filters is formed in each iteration by reproducing 10% of the previous expressions, creating 45% of expressions by crossover and 45% of expressions by mutations of previous filters. In order to select the fittest individuals for reproduction, crossover, and mutation, we use the tournament selection method [BLPW14]. In tournament selection, 8 individuals are randomly selected from the set of expressions and the winner, the expression with the lowest fitness value, is used as a parent for the generation of new expressions.

5. Results

We conducted several experiments with the presented genetic programming framework. We used 200 iterations in each experiment because after 200 iterations the decrease of RelMSE was negligible. The average duration of genetic programming optimization with 6 training scenes was 3 days on a computer with a 4GHz CPU and NVIDIA GeForce GTX 980Ti GPU. We implemented the filtering and calculation of RelMSE on the GPU using CUDA.

We identified one high-quality filtering expression in the results of our experiments. The resulting filter is shown in Equations 2 to 5. In order to validate the capability of this filter to effectively reduce Monte Carlo noise, we compared it to the state of the art learning-based filter (LBF) [KBS15] and to the cross-bilateral filter

(CB) with constant feature weights in three testing scenes (Figure 1 and Figure 3). Evaluation with the third scene can be found in supplementary files. The Monte Carlo noise filter, resulting from our experiments, is defined by the following equations:

$$w(i,j) = e^{-\frac{||f_i^{wp} - f_j^{wp}||_{max}}{0.05}} \cdot e^{power_1} \tag{2}$$

$$power_1 = -\left(\frac{||p_i - p_j||_{L1} + 2 + e^{sinc\left(||f_i^{wp} - f_j^{wp}||_{L2}^{0.002}\right)}}{2^{asin\left(\text{biweight}\left(\left(\left(\frac{||f_i^n - f_j^n||_{L2}}{2}\right)^{power_2}\right)^{power_3}\right)\right)}} \right) \tag{3}$$

$$power_2 = e^{\text{biweight}\left(||c_i - c_j||_{max} 2^{3 \cdot \text{sinc}\left(||f_i^{di} - f_j^{di}||_{L2}\right)}\right)} \tag{4}$$

$$power_3 = e^{-\frac{||f_i^t - f_j^t||_{max}^2}{0.01}} \cdot e^{\text{tricube}\left(||f_i^{di} - f_j^{di}||_{L2}\right) \cdot \text{mitchell}(var^{wp})} \tag{5}$$

f_i^{wp} represents the world position feature at i-th pixel. var^{wp} stands for variance of this feature. *sinc*, *tricube*, *biweight* and *mitchell* are filters from the codebook. $||f_i - f_j||_{L2}$ is Euclidean distance, $||f_i - f_j||_{max}$ is Chebyshev distance and $||f_i - f_j||_{L1}$ is Manhattan distance between i-th and j-th feature values. The full list of terms can be found in supplementary materials.

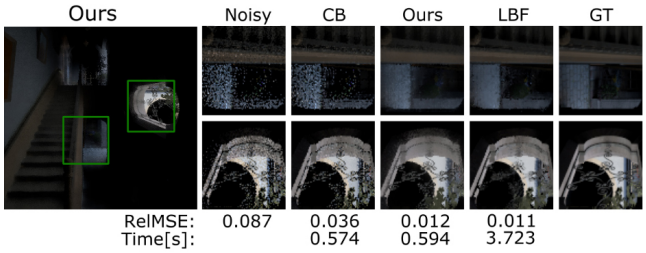

Figure 3: *Comparison of our filter to cross-bilateral (CB) filter and state of the art learning-based filter (LBF) [KBS15]. The noisy input was rendered with 4 samples per pixel.*

Our results show that the presented genetic programming framework is capable of discovering new efficient Monte Carlo noise filters (Figure 1 and Figure 3). In all test scenes our new generated filter achieved higher quality in comparison to the cross-bilateral filter with comparable speed. Additionally, the comparison shows that the quality of our filter is similar to the high-quality LBF filter [KBS15] while achieving the speed one order of magnitude faster. In Figure 1, our filter has lower RelMSE than LBF. However, the LBF method achieves higher visual quality than our filter. This inconsistency suggests that better metrics than RelMSE can be used in the future to measure visual quality. We measured filtering time on a laptop with a 3.3 GHz CPU and GeForce GTX 880M GPU.

Finally, we analyzed data from 20 experiments and we identified scene features the most frequently used by our algorithm for noise reduction: Normal, direct illumination, texture, depth, gradient of secondary texture, and world position (see supplementary files).

6. Conclusion

In this paper, we present a novel method for learning new Monte Carlo noise filters. We employ genetic programming to effectively explore the infinite multidimensional space of analytical filters with the goal to minimize the filtering error. Our results demonstrate that genetic programming is suitable tool for discovery of new efficient filters. Moreover, the analytical nature of resulting filters enables execution times one order of magnitude faster than compared state of the art [KBS15]. Analytical filters have the additional advantage of being human readable and easy to implement on the GPU.

References

[BLPW14] BRADY A., LAWRENCE J., PEERS P., WEIMER W.: genBRDF: Discovering new analytic BRDFs with genetic programming. *ACM Trans. Graph. 33*, 4 (2014), 114:1–114:11. 2, 3

[BRM*16] BITTERLI B., ROUSSELLE F., MOON B., IGLESIAS-GUITIÁN J. A., ADLER D., MITCHELL K., JAROSZ W., NOVÁK J.: Nonlinearly weighted first-order regression for denoising monte carlo renderings. *Computer Graphics Forum 35*, 4 (June 2016). 2

[ETH*09] EGAN K., TSENG Y.-T., HOLZSCHUCH N., DURAND F., RAMAMOORTHI R.: Frequency analysis and sheared reconstruction for rendering motion blur. *ACM Trans. Graph. 28*, 3 (2009), 1–13. 2

[GO12] GASTAL E. S. L., OLIVEIRA M. M.: Adaptive manifolds for real-time high-dimensional filtering. *ACM Trans. Graph. 31*, 4 (2012). 2

[HJW*08] HACHISUKA T., JAROSZ W., WEISTROFFER R. P., DALE K., HUMPHREYS G., ZWICKER M., JENSEN H. W.: Multidimensional adaptive sampling and reconstruction for ray tracing. *ACM Transactions on Graphics 27*, 3 (aug 2008), 1–10. 2

[KBS15] KALANTARI N. K., BAKO S., SEN P.: A machine learning approach for filtering monte carlo noise. *ACM TOG*, 4 (2015). 1, 2, 3, 4

[Koz92] KOZA J. R.: *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Bradford, 1992. 1, 2

[KS06] KENSLER A., SHIRLEY P.: Optimizing ray-triangle intersection via automated search. In *IEEE Symposium on Interactive Ray Tracing* (2006), pp. 33–38. 2

[LAC*11] LEHTINEN J., AILA T., CHEN J., LAINE S., DURAND F.: Temporal light field reconstruction for rendering distribution effects. *ACM Trans. Graph. 30*, 4 (2011), 55:1–55:12. 2

[LALD12] LEHTINEN J., AILA T., LAINE S., DURAND F.: Reconstructing the indirect light field for global illumination. *ACM TOG* (2012). 2

[MCY14] MOON B., CARR N., YOON S.-E.: Adaptive rendering based on weighted local regression. *ACM Trans. Graph. 33*, 5 (2014). 2

[MWRD13] MEHTA S. U., WANG B., RAMAMOORTHI R., DURAND F.: Axis-aligned filtering for interactive physically-based diffuse indirect lighting. *ACM Trans. Graph. 32*, 4 (2013). 2

[ODR09] OVERBECK R. S., DONNER C., RAMAMOORTHI R.: Adaptive wavelet rendering. *ACM Trans. Graph. 28*, 5 (dec 2009). 2

[RKZ11] ROUSSELLE F., KNAUS C., ZWICKER M.: Adaptive sampling and reconstruction using greedy error minimization. In *SIGGRAPH Asia* (New York, NY, USA, 2011), ACM, pp. 1–12. 1, 2, 3

[RKZ12] ROUSSELLE F., KNAUS C., ZWICKER M.: Adaptive rendering with non-local means filtering. *ACM Trans. Graph. 31*, 6 (2012). 1, 2

[SD11] SEN P., DARABI S.: On filtering the noise from the random parameters in monte carlo rendering. *ACM Trans. Graph.* (2011). 1, 2

[YMRD15] YAN L.-Q., MEHTA S. U., RAMAMOORTHI R., DURAND F.: Fast 4D sheared filtering for interactive rendering of distribution effects. *ACM Trans. Graph. 35*, 1 (Dec. 2015), 7:1–7:13. 2

[ZJL*15] ZWICKER M., JAROSZ W., LEHTINEN J., MOON B., RAMAMOORTHI R., ROUSSELLE F., SEN P., SOLER C., YOON S. E.: Recent advances in adaptive sampling and reconstruction for monte carlo rendering. *Computer Graphics Forum* (2015). 1, 2