Coherent Metropolis Light Transport on the GPU using Speculative Mutations

Martin Schmidt Oleg Lobachev Michael Guthe
University Bayreuth,
AI5: Visual Computing
Universitätsstr. 30
95447 Bayreuth, Germany
martin.schmidt@uni-bayreuth.de oleg.lobachev@uni-bayreuth.de michael.guthe@uni-bayreuth.de

ABSTRACT

The Metropolis Light Transport algorithm generates physically based images with superior image quality than classical ray tracing. Although it is trivially parallelizable on GPUs by running *N* MLTs, the performance on current graphics hardware is below par. One of the main problems is the set of incoherent paths due to the independent Markov chains. Since each MLT generates full paths and mutates them sequentially, we construct totally incoherent rays which in negatively affects the performance on the GPU. By using a novel speculative variant of the Metropolis algorithm we increase the similarity of paths and achieve higher coherence. This decreases the computation time significantly. Further, we improve memory access by optimizing the data layout to better utilize coalesced access.

Keywords
global illumination, parallel algorithms, markov chain monte carlo

1 INTRODUCTION

Todayâs standards in generation of high-quality images are based on ray tracing methods. Classical ray tracing [Kaj86] and its extensions have the ability to generate high-quality images with physically correct lighting and shading. Due to the algorithms nature, only visible results are calculated. In contrast to brute-force rasterization approaches, Ray tracing depends only logarithmically on scene complexity [WPS$^+$03].

The emergence of massively parallel computing devices with commodity graphics hardware has led to increased research in the field of real-time raytracing. Modern GPUs can handle several hundred threads simultaneously. Since several years, the performance of commodity hardware (e.g. Desktop PCs) is suitable enough for real-time raytracing approaches [RSH05].

Extending classical ray tracing, the path tracing approach leads to significant improvements of visible image quality due to its physically-based rendering approach. Especially bi-directional path tracing and the

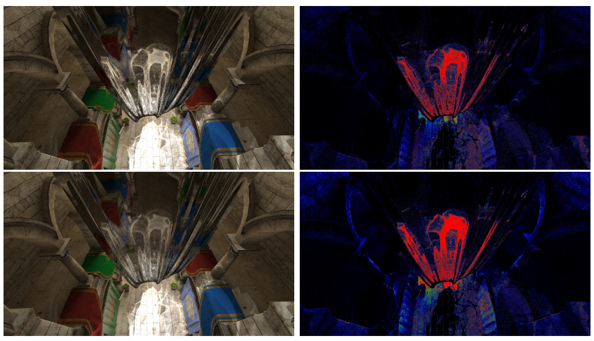
Figure 1: Equal time comparison (10 minutes) of our speculative Metropolis Light transport (top) with a naive parallelization (bottom) running on a GeForce GTX Titan. The error images denote high errors in red, medium in green and low in blue. Our approach clearly improves the convergence rate and reduces the overall error. The image was generated with a speculative tree depth of three.

Metropolis Light Transport (*MLT*) algorithm have build upon the ray tracing idea [Vea97, VG97]. These techniques behave highly parallel due to the independent nature of simultaneously traceable rays [PH10].

We show that the one of the main problems of *MLT* on the GPU, the highly incoherent path samples, can be mitigated by a different parallelization strategy. In addition to performing *N* independent *MLTs*, we propose a speculative mutation algorithm. This reduces the overall incoherence of candidate paths and leads to a reduced divergence and better memory access patterns inside the warps and therefore a higher overall performance.

Figure 1 shows an equal time comparison on a GeForce GTX Titan after 10 minutes with a resolution of 1920×1080. For the speculative MLT, a tree depth of 3 was used. The overall convergence rate is improved not only by the fact that more mutations per pixel are performed, but also since each Markov process performs three steps in a single iteration.

The main contribution of this paper is an efficient parallelization of the *MLT* algorithm on the GPU using CUDA. In the following, we review existing work in the area of GPU-based real-time ray and path tracing (chapter 2), give a short recap on classical *MLT* (chapter 3), describe how the *MLT* can be parallelized using speculative mutations (chapter 4 and 5), show results from our experiments (chapter 6) and give a short outlook on limitations and further possible improvements (chapter 7).

2 RELATED WORK

With the advent of programmable graphics chips, global illumination algorithms were implemented on the GPU. The first GPU-based path tracer was published in 2002 by Purcell et al. [PBMH02]. Much work has been done since these times, including topics like efficient ray traversal [AL09, AK10, Gut14] and ray compaction or sorting [GL09].

Implementing the path tracing algorithm on the GPU has lead to new problems. Since path tracing, Bi-Directional Path Tracing and the Metropolis Light Transport (*MLT*) make use of Monte Carlo sampling methods, their behavior is stochastic in nature. Stochastic sampling lets rays terminate after different path lengths and therefore leads to incoherencies in the workload of each SM on the GPU. This was partially solved by restarting terminated rays [NHD10]. Incoherent branching and memory access lead to reduced performance on modern GPU hardware [ALK12].

Segovia et al. analyzed possibilities to adapt the *MLT* algorithm for SIMD execution on general purpose CPUs [SIP07]. They found out that proposing multiple sub path mutations at once during the mutation step increased parallel SIMD execution feasibility. They implemented the Multiply-Try Metropolis algorithm (*MTM*) [LLW00] to generate a bunch of m mutation candidates during the mutation step for a given MLT sample. These candidates form sub paths for the sampled MLT path that are highly coherent. On the GPU this might also look promising since tracing coherent rays better utilizes the current hardware architecture [ALK12]. The time required per mutation however doubles due to the additional reference set. In addition, each of these paths only contributes a smaller fraction to the image, i.e. $\frac{1}{m}$ on average.

The traditional *MLT* implementation, described by Veach [Vea97, VG97], directly mutates the path vertices. While this might seem straightforward, the implementation is very complicated. Therefore, Kelemen et al. [KSKAC02] proposed to define the Markov process in terms of the random numbers that would be used in bi-directional path tracing. Despite importance sampling, this approach produces images with slightly lower quality than the original MLT. An alternative strategy is described by Hachisuka et al. [HKD14]. By combining *Markov Chain Monte Carlo (MCMC)* sampling with *Multiple Importance Sampling (MIS)*, they further decrease variance and achieve results of similar or even better quality than the original MLT. In our work, we use this combination as it has several advantages due to the way it generates samples.

3 METROPOLIS LIGHT TRANSPORT

The original *Metropolis Light Transport* algorithm [Vea97] extends the idea behind *bi-directional path tracing* (BDPT). It connects paths between light sources and the virtual camera lens to calculate the energy that flows between the light source and the objects and eventually reaches the virtual sensor. While this approach is similar to BDPT, the way paths are generated is different.

MLT first generates a path $\bar{x}$ that starts at the light source. The path then is extended by a series of vertices $x_0, x_1, \dots, x_k$ for a length $k \geq 1$ [VG97]. Each vertex lies on an arbitrary surface and receives energy from the light source through the path. The exact direction for a new path segment is calculated using Monte Carlo sampling. Once a path that successfully connects the light source and the virtual sensor is found, it serves as a starting point for random path mutation.

During path mutation, the algorithm generates a *Markov chain* of path mutations $\overline{X}_0, \overline{X}_1, \dots, \overline{X}_i$. Each mutation $\overline{X}_i$ is the result of a random walk permutation of one of the vertices of the direct predecessor $\overline{X}_{i-1}$. The mutation strategy only uses the direct predecessor as a start for the next mutation, ignoring all earlier mutations. Each new mutation is the result of a *Metropolis-Hastings* sampling in local path space.

For each mutation $\overline{X}'_i$, an *acceptance probability* function $a(\bar{y}|\bar{x})$ evaluates the chance that $\bar{y}$ will be $\overline{X}'_i$ if $\bar{x} = \overline{X}_{i-1}$. If $\overline{X}'_i$ is accepted, it becomes $\overline{X}_i$, otherwise $\overline{X}_{i-1}$ will be kept.

After the mutation has finished (either with acceptance or not), the contribution of the new path is calculated and added to the corresponding pixel in the *image plane*. Each pixel is sampled by a number of n mutations, with example values of 250 [VG97].

The main advantage of the MLT is the local behavior of the mutations. Once a path with a certain contribution has been sampled, each mutation will at first be in the neighborhood of the original path. This increases performance and quality for problematic scenes where light has to travel mostly indirect, for example the Sponza scene where the light source is outside the atrium and only can enter through the open ceiling.

When trying to implement a parallel MLT algorithm, the main problem of the classical MLT is the fact that it randomly walks through path space. So when running N MLTs in parallel, they usually sample completely different areas of the path space at each point in time. This is especially problematic for GPU-based implementations as coherent processing is critical to maintain high performance [AL09].

3.1 Primary Sampling Space MLT

Kelemen et al. [KSKAC02] proposed a novel mutation strategy for MLT path mutation that operates in the *primary sampling space* (PSSMLT). The basis is a bi-directional path tracer for which the random numbers are generated by a Markov chain. Thus the state itself is a set of random numbers that were used to generate the path. This can be seen as a single point $\overline{u}_i$ in a high dimensional – or possible infinitely dimensional – space. The mapping is shown in Figure 2.

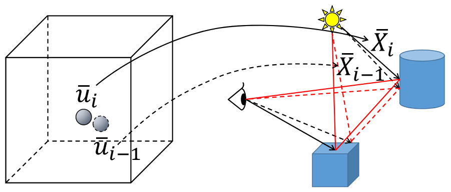

Figure 2: Primary sampling space MLT. Every multi-dimensional sampling point $\overline{u}_i$ is mapped to a path $\overline{X}_i$. Similar points map to similar paths.

Mutations are thus simply local or global movements of this point, where large changes – i.e. generating new random numbers – correspond to the original BDPT. This has three advantages. First, any multi-dimensional random number produces a valid path, where directly changing the path vertices often produces invalid paths. Second, similar sample points produce similar paths which can be used to control the magnitude of path change. And finally, the mutations can easily be made symmetric which removes one of the more complicated terms for computing the acceptance probability.

While all these significantly ease the implementation and reduce the memory that is needed to store the path information, the resulting quality is slightly lower than that of the original MLT. One of the reasons is that BDPT connects the eye and light path at all vertices. In combination with the fact that very often the vertices at the end of the eye and light subpath cannot be connected, this means that the same contribution would have been made by a shorter path that had required less computation time.

3.2 Multiplexed MLT

An improvement of this approach is the *Multiplexed MLT* (MMLT), where the mapping from the random numbers to path space is not unique. Instead of connecting all vertices, only the end points of the eye and light path are connected. For a fixed path length of k, there are thus $k+2$ possible mappings of the random numbers to a path. The eye path length needs to be chosen between 0 to $k+1$ vertices and the light path contains the remaining vertices. This is done using a tempering parameter t that is also selected using the same Markov chain as for the random numbers. The mapping from $(\overline{u},t)_i$ to $\overline{X}_i$ is shown in Figure 3.

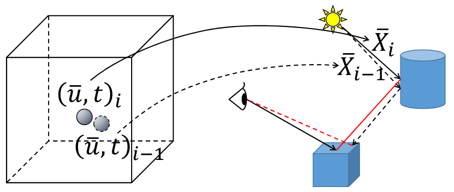

Figure 3: Multiplexed MLT. An additional tempering parameter t is used to determine the length of the eye and light subpath. Only the end points are connected.

In contrast to the PSSMLT, every edge that was traced is used to calculate the contribution. If the end points cannot be connected, e.g. because one of the surface normals points away from the connecting edge, the path contribution becomes zero. This means that such a path is never accepted and the MLT continues its search from the previous path $\overline{X}_{i-1}$ that had a contribution. The same holds for a path with occluded connecting edge. Thus mostly paths where every edge transports energy are sampled. This means that the number of unnecessarily traced rays is reduced compared to PSSMLT. In the end, the method produces results that are competitive with the original MLT with respect to peak signal noise ratio. The simple mapping from multi-dimensional points to valid paths has however a significant advantage as we will discuss in the following.

3.3 Problem Statement

Both of these approaches share the property that similar random numbers generate similar paths. Thus the coherence between neighboring threads that handle different Markov chains can be increased if they are based on similar random numbers. Simply using N independent Markov chains however leads to samples that are distributed throughout the sampling space.

One possibility would be to sort the paths according to their random numbers but this adds a significant overhead that is not easily alleviated by the improved coherency. Using Multiple-Try mutations also generates similar paths but adds the overhead of generating a reference set and reduces the contribution of each path. If m tries are generated, the contribution of each is expected to be $\frac{1}{m}$. Another possibility would be to generate a set of mutations from the current state and successively test them until the first one is accepted [BJB10]. While this reject chain (RC) sampling is a good strategy for Markov Chains with low acceptance probability, all MLT variants try to achieve an acceptance rate close to 1.

If we use a PSSMLT or MMLT, the mutation itself does not depend on the actual path but only on the random numbers that define it. Thus we can generate more than a single mutation – and their corresponding paths – in parallel. Instead of only sampling the reject chain, we can sample all possible paths up to a depth of d.

4 SPECULATIVE MLT

The main idea behind the *speculative MLT* (SMLT) is to simultaneously and speculatively evaluate possible mutations from a candidate set. To this end, we need to perform three steps: First, we generate the sampling points in primary space. Then we evaluate the corresponding paths. Finally, we accumulate the contributions and choose the final candidate.

Performing all possible mutations up to a given depth d produces a binary tree of candidates. As global mutations completely change the sampling points in primary space, we only allow them as first mutation of the local tree. All other mutations are always local ones that produce similar paths. Details on the parallel implementation are discussed in section 5.1.

Then we trace the paths to compute their transported energy, probability and weight as discussed in [HKD14]. From this we can compute the local acceptance probability $a(\overline{X}_b|\overline{X}_a)$ for each mutation from $\overline{X}_a$ to $\overline{X}_b$. For a single mutation step starting at $\overline{X}_{i-1}$, we then have the following iteration:

$$\overline{X}_i = \begin{cases} \overline{X}'_i & : \quad a(\overline{X}'_i|\overline{X}_{i-1}) \\ \overline{X}_{i-1} & : \quad 1-a(\overline{X}'_i|\overline{X}_{i-1}) \end{cases} \tag{1}$$

When extending the local mutation tree to a depth of 2, we also need to consider the two possible mutations from $\overline{X}_{i-1}$ to $\overline{X}^1_{i+1}$ and from $\overline{X}'_i$ to $\overline{X}^2_{i+1}$. If a_1 denotes the acceptance probability from $\overline{X}_{i-1}$ to $\overline{X}_i$, a^2_1 from $\overline{X}_{i-1}$ to $\overline{X}^1_{i+1}$ and a^2_2 from $\overline{X}'_i$ to $\overline{X}^2_{i+1}$, we can write the total acceptance probabilities as:

$$\begin{aligned} p(\overline{X}_{i-1}) &= (1-a_1)\cdot(1-a^2_1) \\ p(\overline{X}^1_{i+1}) &= (1-a_1)\cdot a^2_1 \\ p(\overline{X}_i) &= a_1\cdot(1-a^2_2) \\ p(\overline{X}^2_{i+1}) &= a_1\cdot a^2_2 \end{aligned} \tag{2}$$

Note that these always sum up to $\Sigma p = 1$. Equation 2 is illustrated in Figure 4. For larger trees, the probability for each candidate is the product of all accept/reject probabilities from the root down to the leaf level.

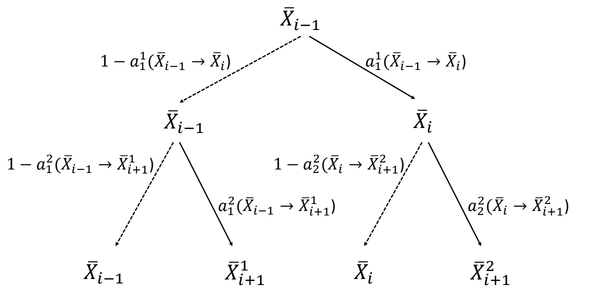
$\bar{X}_{i-1}$
$1-a^1_1(\bar{X}_{i-1}\to\bar{X}_i)$
$a^1_1(\bar{X}_{i-1}\to\bar{X}_i)$
$\bar{X}_{i-1}$
$\bar{X}_i$
$1-a^2_1(\bar{X}_{i-1}\to\bar{X}^1_{i+1})$
$a^2_1(\bar{X}_{i-1}\to\bar{X}^1_{i+1})$
$1-a^2_2(\bar{X}_i\to\bar{X}^2_{i+1})$
$a^2_2(\bar{X}_i\to\bar{X}^2_{i+1})$
$\bar{X}_{i-1}$
$\bar{X}^1_{i+1}$
$\bar{X}_i$
$\bar{X}^2_{i+1}$

Figure 4: Mutation tree and acceptance probabilities for a mutation depth of $d = 2$.

Although we have a candidate set of 2^d samples, we only need to trace the path for $2^d - 1$ of them. $\overline{X}_{i-1}$ is the first-chance rejection and therefore still the same path as the result from the last iteration. By increasing the size of the candidate set, we perform d iterations in parallel on a set of $\sim \frac{N}{2^d-1}$ MLTs. So in addition to more efficient tracing, we also expect a lower start-up bias and faster convergence to the stationary distribution.

4.1 Variance reduction

Similar to previous methods [VG97, HKD14], we want to minimize the variance of the generated image. Therefore, we accumulate the expectation value of all candidates instead of the chosen path only.

In contrast to computing the acceptance probability, we need however not only consider the leaf level of the tree as this would mean to skip all iterations except the last one. Instead, we calculate the contribution at each level of the tree, except the root node which was already accumulated in the last step. In our example with a depth of 2, the contribution weight w for each path is:

$$\begin{aligned} w(\overline{X}_{i-1}) &= (1-a_1)\cdot(2-a^2_1) \\ w(\overline{X}^1_{i+1}) &= (1-a_1)\cdot a^2_1 \\ w(\overline{X}_i) &= a_1\cdot(2-a^2_2) \\ w(\overline{X}^2_{i+1}) &= a_1\cdot a^2_2 \end{aligned} \tag{3}$$

Node that these always sum up to $\Sigma w = d$, which is 2 in this example. Like in this example, the equations for deeper trees are similar to the acceptance rates. The only difference is that – with the exception of a_1 as discussed before – all $(1-a^i_j)$ become $(2-a^i_j)$.

5 IMPLEMENTATION WITH CUDA

For the implementation of the speculative MLT, we extended our existing MMLT implementation in CUDA.

The general process per iteration can be subdivided into the following parts: First, we generate the candidate mutations for the given depth d. Then we trace all new paths. Finally, we compute the contributions and select the surviving paths.

5.1 Mutation

The mutation step computes one candidate per thread. From each current path p, we generate a set of $2^d - 1$ candidates. Each thread loops over the variables of p in the primary sampling space. For each variable i, we first load the corresponding one from p. By arranging the variables in the path buffer at position $p + i \cdot N$, we access them with a stride of 1 and partial broadcasts.

Then each thread generates a random number representing the last mutation of variable i. To apply the mutation to all relevant candidates, this number is stored in shared memory. Starting from the first mutation, they are applied to all relevant candidates per level using the original mutation strategy [KSKAC02]. For all mutations except the last one, the random number is fetched from shared memory. Note that this is similar to a reduction using a binary tree in shared memory. The only difference is that the accumulation is performed towards the leaves and not towards the root. Figure 5 shows this process for an example depth of 2. Finally, the new sampling space variables are again stored with stride 1 access in the larger candidate buffer.

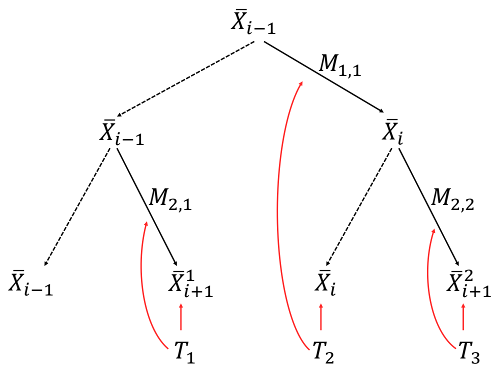

Figure 5: 3 threads are working on 3 mutations; mutation depth of $d = 2$.

5.2 Tracing

After generating the candidates in primary sampling space, we trace the paths using a ray scheduler based on the wavefront path tracer [LKA13]. First all eye and light segments are stored in a ray buffer. Then the intersection points for these rays are determined using a highly optimized trace kernel [Gut14]. The BRDF is evaluated at the hit points and secondary rays are constructed based on the stored *random numbers* in the candidate set. Finally, the shadow rays to connect the path ends are constructed and again traced using the same trace kernel.

While tracing will be divergent even for coherent paths, ray construction is mostly non-divergent (except for rays that exited the scene). Here again, the memory layout of the candidate set leads to a coalesced stride 1 access into the candidate buffer.

Due to the ray restarting of the trace kernel, the performance will gradually increase with the number of coherent rays. However, it will level once it reaches the warp size of (currently) 32.

5.3 Selection

For selection, we could again start one thread per candidate and coordinate the work over shared memory. As we still have a high amount of parallel MLTs, it is however more efficient to handle each selected path p with its own thread.

While reading the sample state now has a stride of $2^d - 1$, the problem can be alleviated for the path contribution. Each candidate path c stores *RGB* values for the transported energy multiplied with the weight from the balance heuristic [HKD14] and a pixel index for each pixel it contributes to. By combining this data into a single `float4`, we only need a single memory access and loose fewer bandwidth. The pixel index is then accessed using the *intrinsic* functions `__int_as_float` and `__float_as_int`.

6 RESULTS

We tested our implementation on an *Intel Core i7-3720* quadcore CPU with 16 GB of RAM, paired with an *nVidia GeForce GTX Titan* with 6 GB video memory. As test cases we chose three scenes listed in table 1 to evaluate different light transport scenarios.

Scene	# Triangles	# Vertices
Sponza	279,157	193,300
Sibenik	75,284	83,490
Conference	331,179	216,862

Table 1: Number of triangles and vertices per mesh.

Figure 6 shows the ground truth images of the views we used for the evaluation. These images were generated using a bi-directional path tracer with 10 million samples per pixel with a maximum of 10 bounces and the pseudo-random Halton sequence. We added a highly specular banner to the Sponza scene (cosine lobe with an exponent of 10^5) to produce caustics and reflected caustics. These are especially visible in view 2. The Sibenik cathedral is illuminated by the light shining through the windows only, so only few paths contribute to the image. This is even more the case in the final example, the Conference scene. Here there is only indirect light coming through the sunblind with at least two bounces.

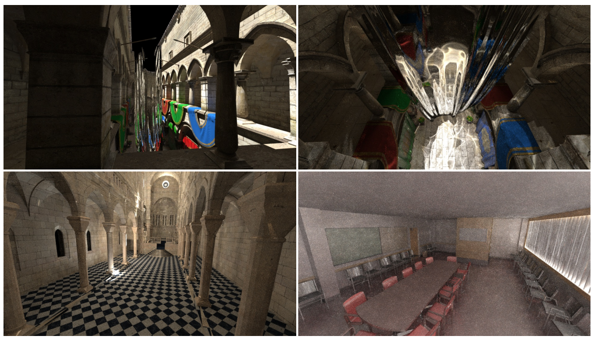

Figure 6: Scenes and views (ground truth images) used in the evaluation. From left to right and top to bottom: Sponza view 1, Sponza view 2, Sibenik and Conference.

Each scene was tested with the same view and a screen resolution of 1920×1080 for all of the methods. We compared the naively parallelized MMLT with our speculative MLT using different mutation depths and the reject chain MLT using different chain lengths. Table 2 shows the number of parallel MLTs N and the total number of parallel mutations. For the *mutate kernel*, we thus launch 1024 blocks with 248 to 256 threads.

	# MLTs (N)	# mutations
naive	$256 \cdot 2^{10}$	$256 \cdot 2^{10}$
S2	$85 \cdot 2^{10}$	$255 \cdot 2^{10}$
S3	$36 \cdot 2^{10}$	$252 \cdot 2^{10}$
S4	$17 \cdot 2^{10}$	$255 \cdot 2^{10}$
S5	$8 \cdot 2^{10}$	$248 \cdot 2^{10}$
S6	$4 \cdot 2^{10}$	$252 \cdot 2^{10}$
RC2	$128 \cdot 2^{10}$	$256 \cdot 2^{10}$
RC4	$64 \cdot 2^{10}$	$256 \cdot 2^{10}$
RC8	$32 \cdot 2^{10}$	$256 \cdot 2^{10}$
RC16	$16 \cdot 2^{10}$	$256 \cdot 2^{10}$
RC32	$8 \cdot 2^{10}$	$256 \cdot 2^{10}$
RC64	$4 \cdot 2^{10}$	$256 \cdot 2^{10}$

Table 2: Number of parallel running MLTs and the number of parallel mutations.

The results show an almost linear speedup with the depth for our speculative parallelization, compared to the naive approach of using independent MLTs. Table 3 and Figure 7 show the linear growth of the performance for both. The main reason for the speedup is the increase in coherence that directly translates into a higher trace performance. Once the set of coherent paths grows beyond the warp size of 32, the performance does not increase any more. This is as expected since packets of at most 32 rays are fetched by the ray tracer [ALK12].

In addition, we compare the peak signal noise ratio of the generated images after rendering for 10 minutes in Table 4. Depending on the scene characteristics, the

	Sponza view 1	Sponza view 2	Sibenik	Conference
naive	14.48	9.80	15.92	8.31
S2	15.61	10.44	16.40	8.90
S3	15.67	10.69	16.86	9.07
S4	15.87	10.86	17.11	9.47
S5	16.33	11.09	17.60	9.77
S6	16.42	11.10	17.63	9.78
RC2	15.49	10.36	16.41	8.68
RC4	15.84	10.98	17.51	9.19
RC8	16.89	11.40	18.18	9.65
RC16	17.66	11.99	18.38	10.19
RC32	17.67	12.39	19.24	10.63
RC64	17.76	12.53	19.36	10.64

Table 3: Million mutations per second for the naive parallelization, the speculative with depth d (Sd) and reject chain with length l (RCl).

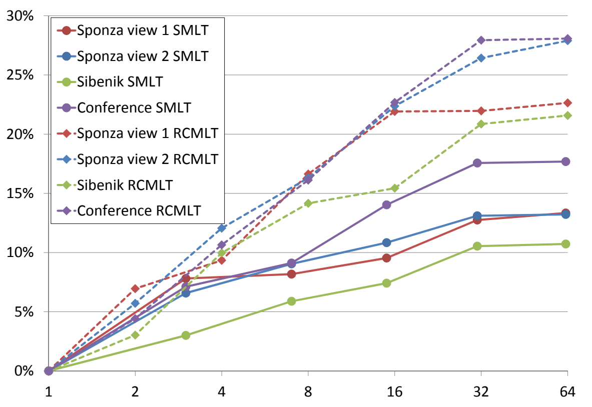

Figure 7: Relative speedup of speculative MLT (SMLT) and reject chain MLT (RCMLT) compared to naively parallelized MMLT; on the x-axis are the sizes of candidate sets.

best results can be achieved with $d = 2$ (Sibenik, Conference) to $d = 3$ (Sponza view 2). With increasing depth, more paths are wasted and the quality gradually drops. This is especially true for simple views, where the acceptance probability is close to 1. In such cases, e.g. view 1 of the Sponza scene, the naive parallelization produces the best results. Note that the reject chain approach never produces better images in the same time than the naive one. This is due to the fact that the contribution of the first additional candidates is almost always below $\frac{1}{4}$ and drops exponentially with further ones.

Figure 8 compares the generated images using the best parallelization for each view with the naive approach. The error and noise are especially reduced in difficult cases. These are the reflected caustic in sponza view 2 and the indirect light illuminating the conference scene. Note that for sponza view 2, neither of them has been able to converge to the stationary distribution yet, so the reflected caustic appears slightly too dark in both images.

	Sponza view 1	Sponza view 2	Sibenik	Conference
naive	**28.61**	17.27	28.50	12.44
S2	28.47	17.90	**28.60**	**12.50**
S3	28.00	**18.10**	27.64	12.20
S4	27.84	17.88	26.19	11.62
S5	27.56	17.72	24.80	11.09
S6	27.17	17.33	23.29	10.38
RC2	28.29	17.09	26.03	11.59
RC4	26.21	16.43	23.13	10.73
RC8	20.50	15.12	20.24	9.90
RC16	18.76	14.35	17.63	9.10
RC32	17.95	13.99	15.91	8.43
RC64	17.27	14.00	15.45	7.93

Table 4: Peak signal noise ratio comparison of different parallelization strategies after rendering 10 minutes. The best is marked in bold.

Figure 9 shows an equal time comparison for speculative tree depths from 1 (naive) to 6. While the error in the reflected caustic is decreasing with higher tree depth, the overall error increases due to the decreasing weights of deeper paths. This clearly shows that deeper trees are suitable for paths that are difficult to sample.

7 CONCLUSION AND LIMITATIONS

We have proposed a novel approach for parallelizing the Metropolis Light Transport algorithm on the GPU. Our approach successfully utilizes the graphics hardware to achieve a substantial speedup compared to naively parallelized MLT. This is mostly accomplished by evaluating paths that are more coherent. This shows significantly better performance on GPUs where divergence in both execution and memory access imposes a severe penalty.

While our approach could be extended to other Metropolis sampling algorithms, it requires that the sample generation and evaluation can be decoupled. This means that it must be possible to generate the candidate $\overline{X}_i'$ without having evaluated the previous sample $\overline{X}_{i-1}$. Therefore, our approach cannot be applied to the original MLT algorithm where the path is mutated directly.

Another problem of our approach is that the contribution of a sample exponentially decreases with the depth. This leads to an optimal depth of 2 or 3 which in turn only generated 3 or 7 coherent samples. On the other hand, paths that are difficult to sample – like reflected caustics – are better handled with a depth of 5 or even 6, i.e. 31 or 63 coherent samples. In the future we therefore plan to evaluate other sampling strategies to generate larger sample sets without reducing the contribution of some paths.

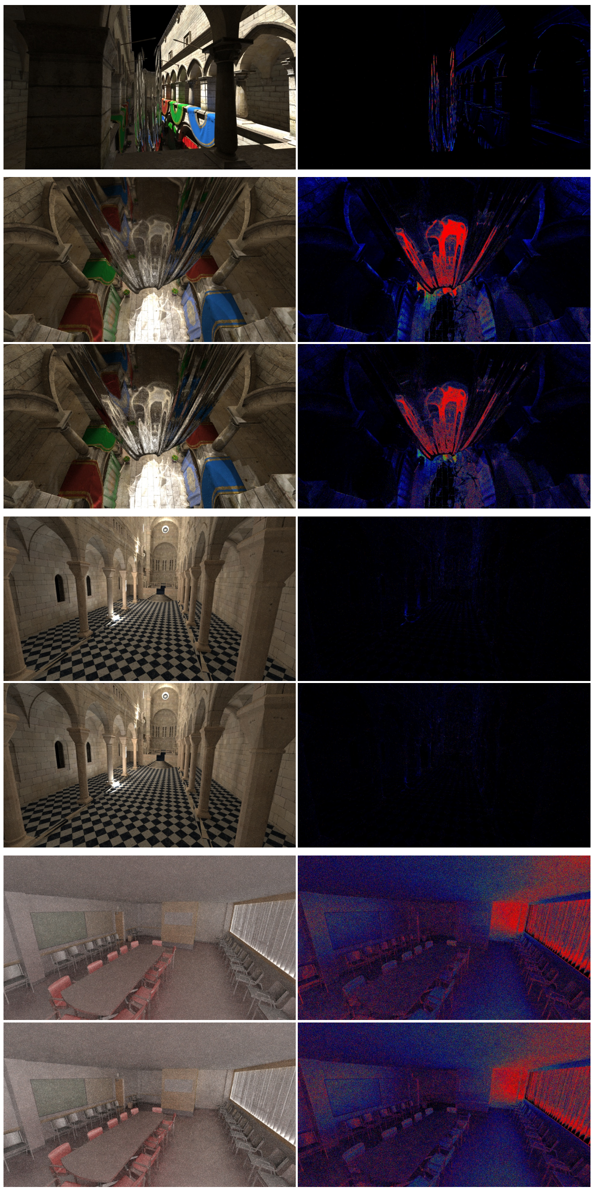

Figure 8: Equal time comparison (10 minutes on a GeForce GTX Titan) using the naive parallelization (upper image) and the best method for each (lower image). The chosen method from top to bottom is: naive, S3, S2 and S2 (c.f. Table 4). The error images show high errors in red, medium in green and low in blue/black.

8 REFERENCES

[AK10] T. Aila and T. Karras. Architecture considerations for tracing incoherent rays. In *Proceedings of High-Performance Graphics 2010*, pages 113–122, 2010.

[AL09] T. Aila and S. Laine. Understanding the efficiency of ray traversal on gpus. In *Proceedings of High-Performance Graphics 2009*, pages 145–149, 2009.

[ALK12] T. Aila, S. Laine, and T. Karras. Under-

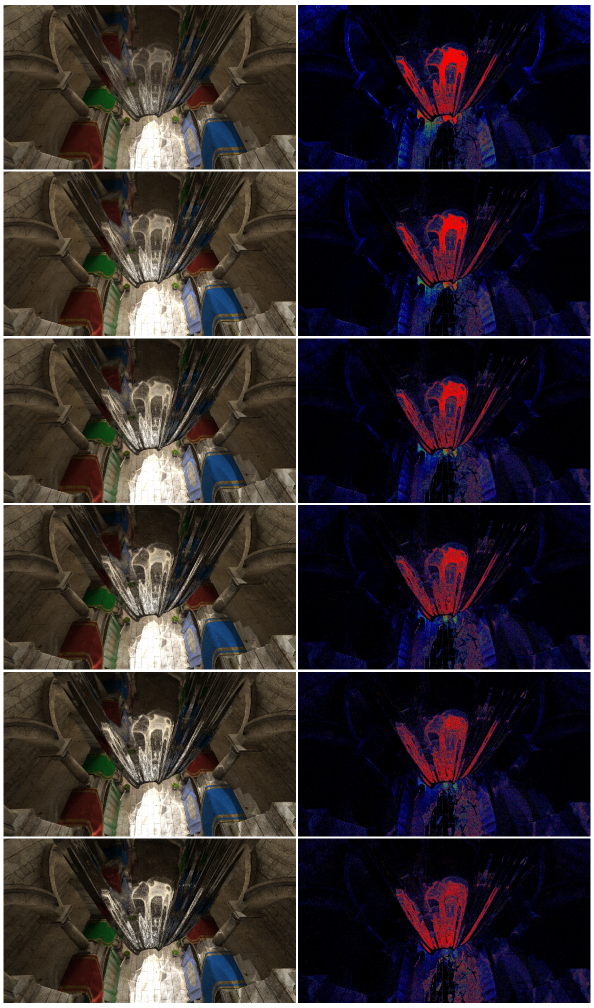
Figure 9: Sponza view 2 with different speculative tree depth d increasing from 1 (naive) to 6. All images were generated in 10 minutes on the GeForce GTX Titan.

standing the efficiency of ray traversal on GPUs – Kepler and Fermi addendum. NVIDIA Technical Report NVR-2012-02, NVIDIA Corporation, June 2012.

[BJB10] J. Byrd, S. Jarvis, and A. Bhalerao. On the parallelisation of mcmc by speculative chain execution. In *Parallel Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE International Symposium on*, pages 1–8, April 2010.

[GL09] K. Garanzha and C. Loop. Fast ray sorting and breadth-first packet traversal for gpu ray tracing. *Computer Graphics Forum*, 29(2):289–298, 2009.

[Gut14] M. Guthe. Latency considerations of depth-first gpu ray tracing. In *Proceedings of Eurographics 2014 - Short Papers*, pages 53–56, 2014.

[HKD14] T. Hachisuka, A. S. Kaplanyan, and C. Dachsbacher. Multiplexed metropolis light transport. *ACM Trans. Graph.*, 33(4):100:1–100:10, July 2014.

[Kaj86] J. T. Kajiya. The rendering equation. In *SIGGRAPH '86 - Proceedings of the 13th annual conference on Computer graphics and interactive techniques*, 1986.

[KSKAC02] C. Kelemen, L. Szirmay-Kalos, G. Antal, and F. Csonka. A simple and robust mutation strategy for the metropolis light transport algorithm. *Computer Graphics Forum*, 21(3):531–540, 2002.

[LKA13] S. Laine, T. Karras, and T. Aila. Megakernels considered harmful: Wavefront path tracing on gpus. In *Proceedings of High-Performance Graphics 2013*, 2013.

[LLW00] J. S. Liu, F. Liang, and W. H. Wong. The multiple-try method and local optimization in metropolis sampling. *Journal of the American Statistical Association*, 95(449):121–134, 2000.

[NHD10] J. Novák, V. Havran, and C. Dachsbacher. Path regeneration for interactive path tracing. *Proceedings of Eurographics 2010 - Short Papers*, pages 61–64, 2010.

[PBMH02] T. J. Purcell, I. Buck, W. R. Mark, and P. Hanrahan. Ray tracing on programmable graphics hardware. In *Proceedings of ACM SIGGRAPH 2002*, pages 703–712, 2002.

[PH10] M. Pharr and G. Humphreys. *Physically Based Rendering, Second Edition: From Theory To Implementation*. Morgan Kaufmann Publishers Inc., 2nd edition, 2010.

[RSH05] A. Reshetov, A. Soupikov, and J. Hurley. Multi-level ray tracing algorithm. *ACM Transactions on Graphics*, 24(3):1176–1185, July 2005.

[SIP07] B. Segovia, J.-C. Iehl, and B. Péroche. Coherent Metropolis Light Transport with Multiple-Try Mutations. Technical report, LIRIS UMR 5205 CNRS/INSA de Lyon/Université Claude Bernard Lyon 1/Université Lumière Lyon 2/École Centrale de Lyon, April 2007.

[Vea97] E. Veach. *Robust Monte Carlo methods for Light Transport Simulation*. PhD thesis, Standford University, 1997.

[VG97] E. Veach and L. J. Guibas. Metropolis light transport. In *SIGGRAPH '97 - Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '97, pages 65–76, 1997.

[WPS$^+$03] I. Wald, T. J. Purcell, J. Schmittler, C. Benthin, and P. Slusallek. Realtime ray tracing and its use for interactive global illumination. Eurographics State of the Art Reports, 2003.