Supplemental document for selective guided sampling with complete light transport paths

FLORIAN REIBOLD, JOHANNES HANIKA, ALISA JUNG, AND CARSTEN DACHSBACHER, KIT, Germany

Fig. 1. We show results of our method in the DiningRoom (left) and Pool (right) scene with a high sample count to illustrate that our method converges in the limit and can produce clean results without visible artifacts. The comparison to PT (bottom row) shows that no overfitting is taking place, which would potentially lead to infinite variance or missing contributions. No outlier removal (DBOR) or denoising was used on these images.

1 EXTRA COMPARISONS

In Figure 1 we show more converged results of our method in the DiningRoom (left) and Pool (right) scene to illustrate that our method converges in the limit and can produce clean results without visible artifacts. See Figure 2 for frames of a progressive rendering of our algorithm.

In Figure 4 we show a closeup of the DiningRoom scene focusing on the caustic and reflected caustic. Our implementation can be used with any path sampler for which we can compute the PDF in closed form. While path tracing is a reasonable choice, we also demonstrate guiding in combination with bidirectional path tracing (BDPT) [Lafortune and Willems 1993; Veach and Guibas 1994] as the unguided sampler. As expected, BDPT fails to resolve the reflected caustics. In addition, the relatively large and bright exterior light source attracts most of the samples for the light tracing pass, making this technique inefficient. The guided sampling versions both used a learning phase of 30s out of 5min total render time (in both cases about 17 iterations, resulting in approx. 5700 guide paths). We also compare to Markov chain Monte Carlo methods which can be used to render such (reflected) caustics efficiently. However, they introduce correlated sampling artifacts throughout the image, not only in the difficult regions. Note that both HSLT [Hanika et al. 2015; Kaplanyan et al. 2014] and KMLT [Kelemen et al. 2002] in this figure use unidirectional path tracing only. In Figure 5 we show the same experiment but without the area light (spotlight only). Methods using light tracing work much better in this setting because the area light from the scene in the paper does not take up many unimportant samples anymore.

The first scene in Figure 6 shows a hair strand that is partly inside a glass sphere. This scene is difficult for many rendering algorithms: BDPT fails at sampling important paths as the glass sphere prevents next event estimation to the light source as well as to the camera.

Author's address: Florian Reibold, Johannes Hanika, Alisa Jung, and Carsten Dachsbacher, KIT, Am Fasanengarten 5, Karlsruhe, BW, 76131, Germany, florian.simon@kit.edu.

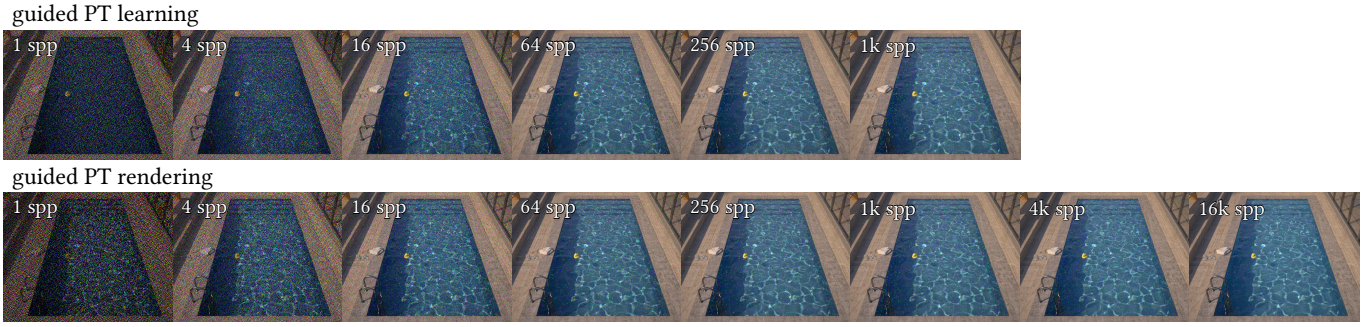


Fig. 2. Some frames of a progressive rendering of our algorithm showing the learning phase (top) and the rendering phase (bottom) where the framebuffer is cleared after learning.

Fig. 3. The cascade of DBOR buffers for the pool scene after 30min of render time using our guided PT (top) and PT (bottom). In this scene outlier removal basically means ignoring the higher levels of the DBOR cascade. As can be seen, the higher levels of the guided PT cascade contain significantly less energy compared to the PT cascade and the caustics are already resolved in the lower levels of the DBOR cascade. For better visibility the i-th cascade level ($i = 0, \ldots 9$) is scaled by 2^i.

Photon mapping struggles in this scene because photon density estimation on hair is difficult: both the approximation of the area as a disk (often used for surfaces) and the volume as a sphere will result in visible errors. Our guiding approach works on hair fibers out of the box, and improves the image quality compared to the unguided version of PT significantly.

The second scene (VeachDoor) in Figure 6 is an example for a scene that is unsuited for our guided sampling in its current form. In this scene all paths are roughly equally hard to sample and there is no isolated lighting effect that guiding could focus on. This means that guiding will try to compute the whole multibounce diffuse light transport which it struggles with due to the curse of dimensionality. In this scene our guided PT therefore degenerates to PT, however, with an increased overhead which results in only half of the samples per pixel per time compared to regular PT.

In Figure 7 we show the rendering results with and without outlier removal for all scenes in the paper as well as the part that was filtered by the outlier removal in a post process.

2 GUIDED SAMPLING VISUALISATION

We used a simple blender visualisation for debugging the guided sampler. In Figure 8 we show a guide path exemplar for two scenes as well as its nearest neighbouring guide paths, Gaussians and some stochastically sampled paths that result from the sampling process.

3 DENSITY-BASED OUTLIER REJECTION

We employ density-based outlier rejection (DBOR) [DeCoro et al. 2010] in two ways: to remove spurious residual outlier samples from the final render, and to detect potential guide paths.

Outlier removal. Outliers can occur in areas where the learning phase did not detect a path contribution yet, but during rendering the unguided sampler created such a rare sample. For higher dimensional problems, it can also occur that the rare sample is on the boundary of a learned area and its PDF is the product of many Gaussian-shaped distributions where the value comes from the long tail with low probability density. These can multiply to extremely low values, resulting in outlier samples. Both indicate that the problem space has not yet been learned sufficiently. Since it may be prohibitive to do so within a certain render budget, it may be useful to remove outliers.

Our implementation follows the cascaded framebuffer scheme as proposed by Zirr et al. [2018], i.e. we do not need to construct and update a kd-tree and the outlier removal is strictly a post-process and as such adjusts to progressive rendering. For an illustration of the buffers, see Figure 3.

Guide path selection. The second application of DBOR is to select which paths to add to the guiding cache. This is in a sense the reverse case of outlier removal: only paths causing high variance in the estimator are admitted to the cache for further exploration

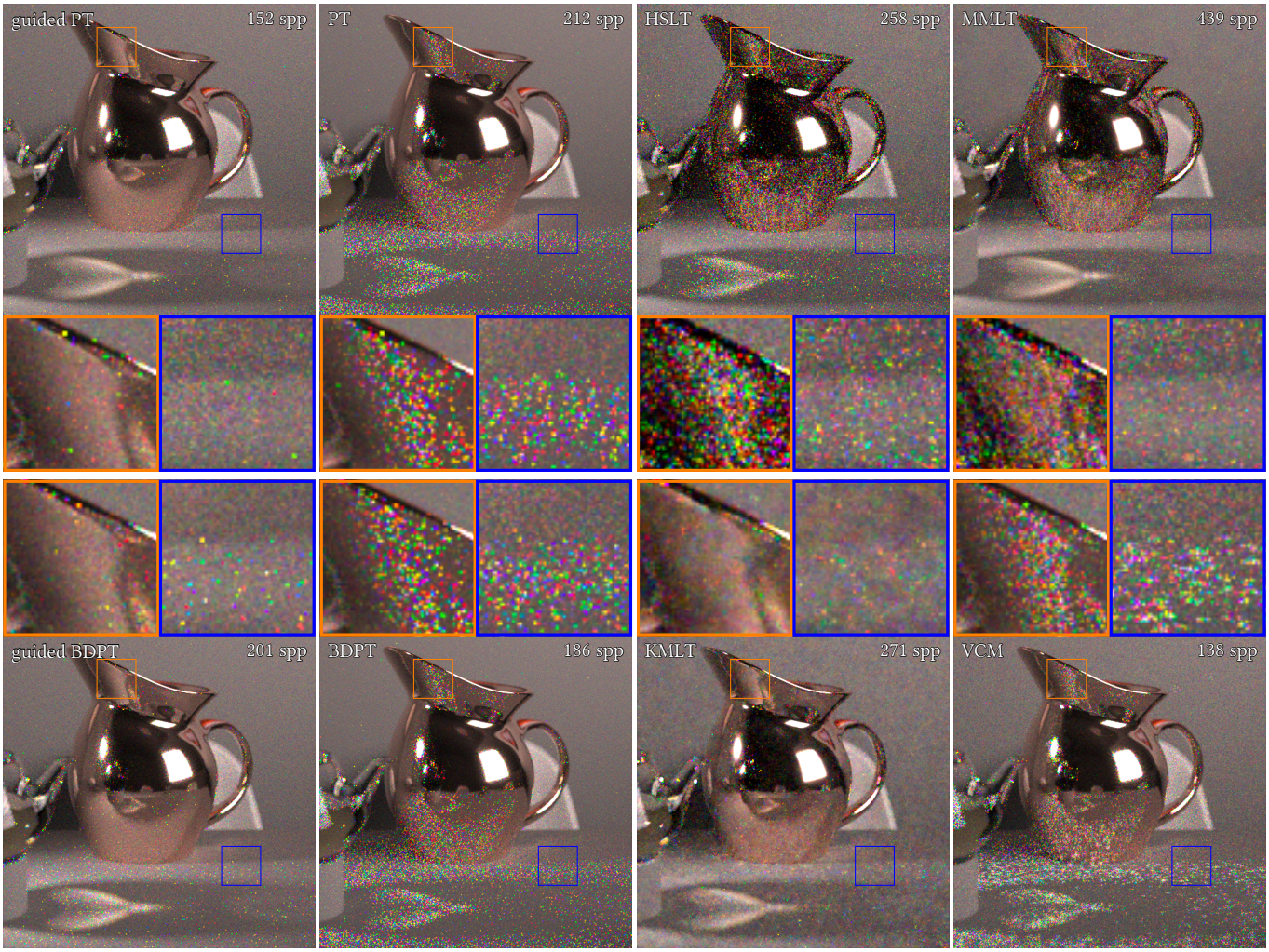


Fig. 4. An equal-time comparison for a closeup view of the DiningRoom. The leftmost column shows guided sampling in conjunction with PT (top) and BDPT (bottom). BDPT and light tracing in general perform very poorly in this scene because the bright area light outside the scene attracts most samples and leaves very few for the prominent spot light which casts the caustic. Vertex connection and merging (VCM) [Georgiev et al. 2012] suffers from the same problem. The selection phase of our guided sampling is able to focus on the caustic and especially the reflected caustic much better. Markov chain-based methods help here, we show three versions: Half vector space light transport (HSLT) [Kaplanyan et al. 2014] with a relatively large step size results in evenly spaced samples but relatively high noise level. Kelemen Metropolis (KMLT) [Kelemen et al. 2002] uses a small step size leading to blotchy appearance, especially in the blue inset or the back wall. Multiplexed Metropolis (MMLT) [Hachisuka et al. 2014] looks more even since it uses larger steps and takes advantage of light tracing, but shows scratch-like structures in the orange inset. We derive step sizes from neighbouring samples instead, leading to smoother results.

in subsequent iterations. For this, we keep a data structure similar to the bilateral grid [Chen et al. 2007]: a set of downsampled framebuffers, one slice for every interval in a logarithmically spaced contribution to the frame buffer. Our implementation reduces the size of the DBOR grid by a factor of two, i.e. four image pixels correspond to one DBOR pixel.

During rendering in the learning phase, every sample is splatted into the DBOR grid using a Blackman-Harris filter with 4×4 pixels support. We splat a constant one for each sample into the two levels in the brightness cascade which are closest to the sample contribution. This means the buffer is used to count the number of samples with similar contribution. During evaluation, the grid is simply accessed by reading out a single pixel for each of the two neighbouring brightness levels of the sample (slicing), which are interpolated to yield a single value. This value is used to count the number of similar samples. Based on this we can decide with a threshold whether this contribution is already sufficiently well sampled or needs to be included as a new guide path.

A note on spectral rendering. Because our rendering engine is spectral, we compute the sample contribution relevant for DBOR as X+Y+Z in the CIE XYZ colour space. This distributes spectral power

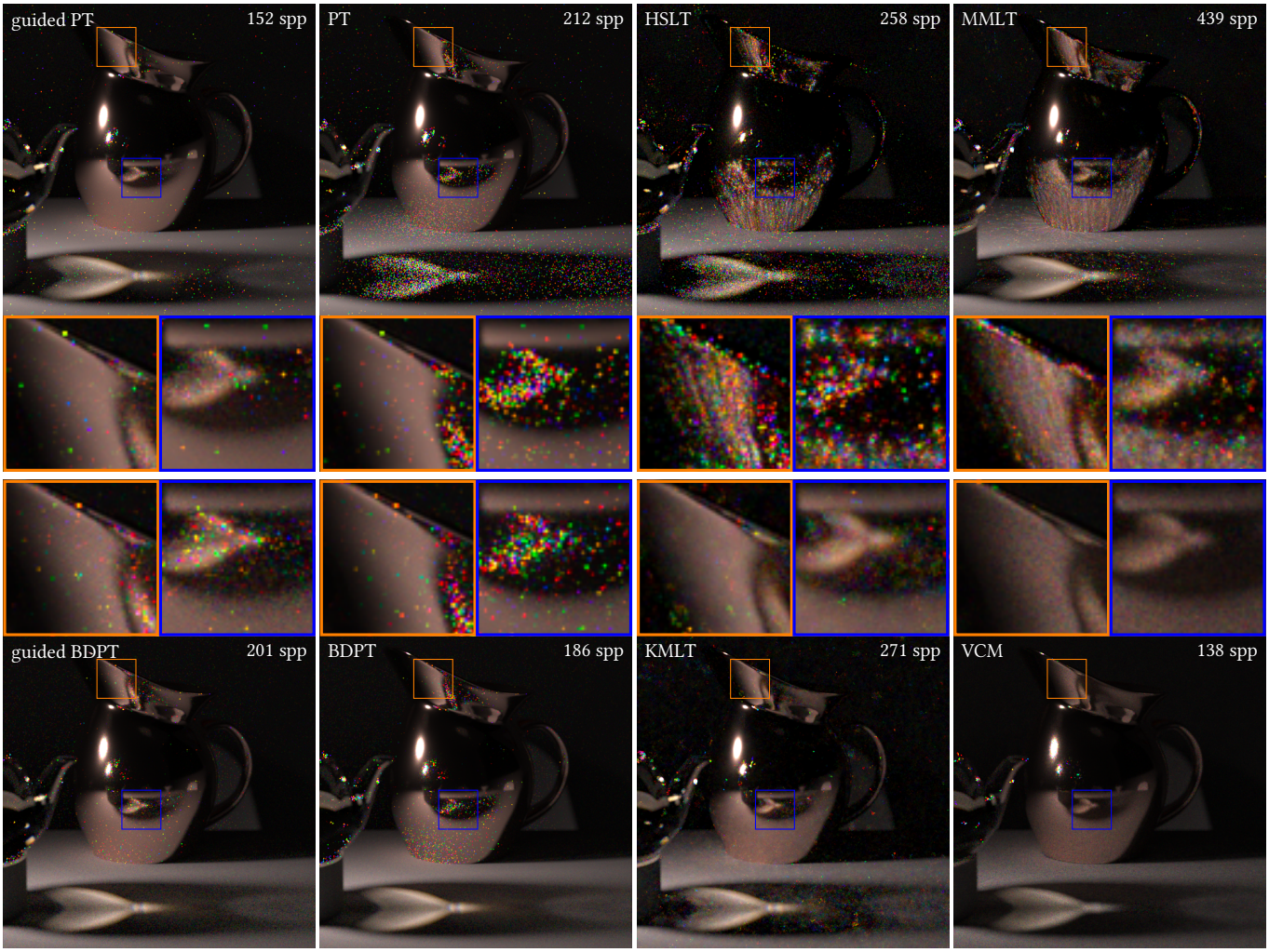


Fig. 5. The same comparison as in Figure 4 but with the exterior light removed, only the spot light remains. This makes light tracing a lot more efficient as can be seen for BDPT and VCM. Note that the Markov-chain based methods (HSLT, KMLT, MMLT) do not perform much differently.

evenly between the colour channels and thus results in much better classification than using an RGB colour space such as rec709 for this purpose.

4 RESULTS WITH AND WITHOUT OUTLIER REMOVAL

In Figure 7 we show the results of path tracing and guided path tracing with and without outlier removal for additional scenes. The images of the guided path tracer have clearly visible but isolated bright outliers. Removing them results in a low noise image.

5 RENDERING ANIMATIONS AND TEMPORAL STABILITY

We tested our guided sampling scheme for temporal stability. The supplemental video shows an intricate caustic from a fluid splash under rotating illumination in the Tumbler scene. This is rendered by (unidirectional) path tracing, guided path tracing, and a variant of guided path tracing that reuses the guiding cache from the last frame of the animation, to explicitly increase temporal stability.

This time-dependent learning is in fact very similar to classic sequential Monte Carlo methods. Because of this, we employ a classic resampling step which starts with the guiding cache of the previous frame. We draw samples from the old sampling distribution, until a certain number of new guide paths is created this way (we use 40% of the input cache size in our implementation). After this step, we discard all paths that came straight from the input and only keep the newly resampled paths, as those faithfully reflect the updated geometric configuration of the scene. This step exploits temporal coherence in a similar way as our learning phase exploits spatial coherence between transport paths. After resampling, we continue the regular learning phase as in the single-frame case. In the supplemental video, we distribute the budget of learning

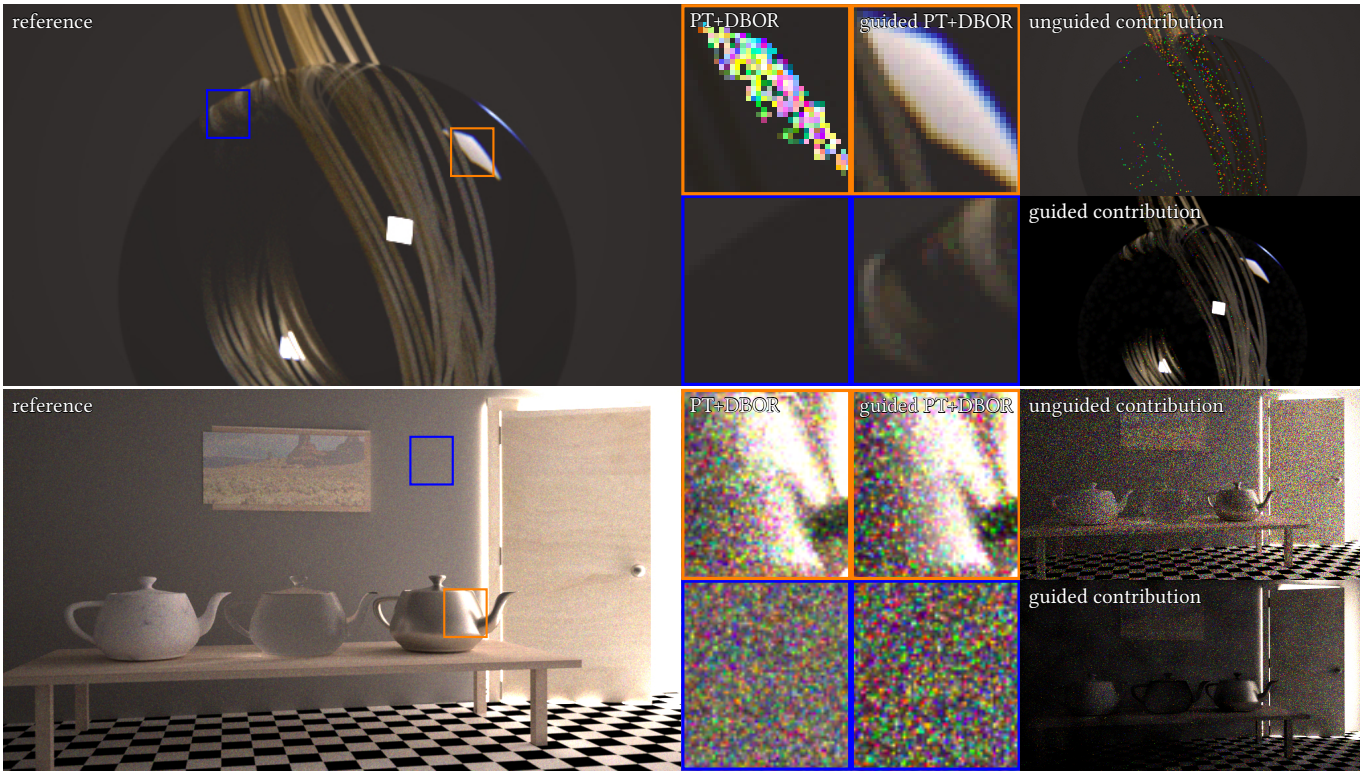


Fig. 6. A hair strand (top) inside a glass sphere is difficult for many rendering algorithms since no direct connection to the camera or the light source is possible. Our guiding approach works on hair fibers out of the box, and improves the image quality compared to the unguided version of PT which for example struggles to sample the indirect reflection of the light source on the glass sphere (top insets). In the VeachDoor scene (bottom) all paths are roughly equally hard to sample and guiding struggles due to the curse of dimensionality. In this scene guided PT degenerates to PT with an 2× overhead. See Table 1 in the paper for details.

iterations between resampling and learning, so the resampling step does not incur extra cost.

The video also shows all three variants (path tracing, independent guiding, and guiding with inter-frame cache reuse) with DBOR outlier removal switched on. It can be seen that independent guiding is stable only in the features that can be detected reliably during the learning phase, and can introduce flickering for more difficult effects such as small glints. The reuse and resampling approach alleviates most of these issues. Note that the Tumbler is not an easy case for any path tracing algorithm, as many small and potentially indirect glints occur on the fine details of the fluid surface.

6 TRIDIAGONAL STRUCTURE AND SPARSE COVARIANCE MATRICES

We employ a Gaussian distribution to sample the next path vertex according to a guide path. For the surface case, this distribution is the 2D conditional of a 4D Gaussian distribution which includes not only the covariance around the next vertex, but also around the current vertex.

This is motivated by the observation that the measurement contribution function in the surface transport case mainly changes with the half vector at path vertices. This half vector affects the main factor of most BSDF models (the evaluation of the distribution of microfacet orientations), while the other factors (Fresnel, shadowing/masking, geometry terms) have less of an impact [Kaplanyan et al. 2014].

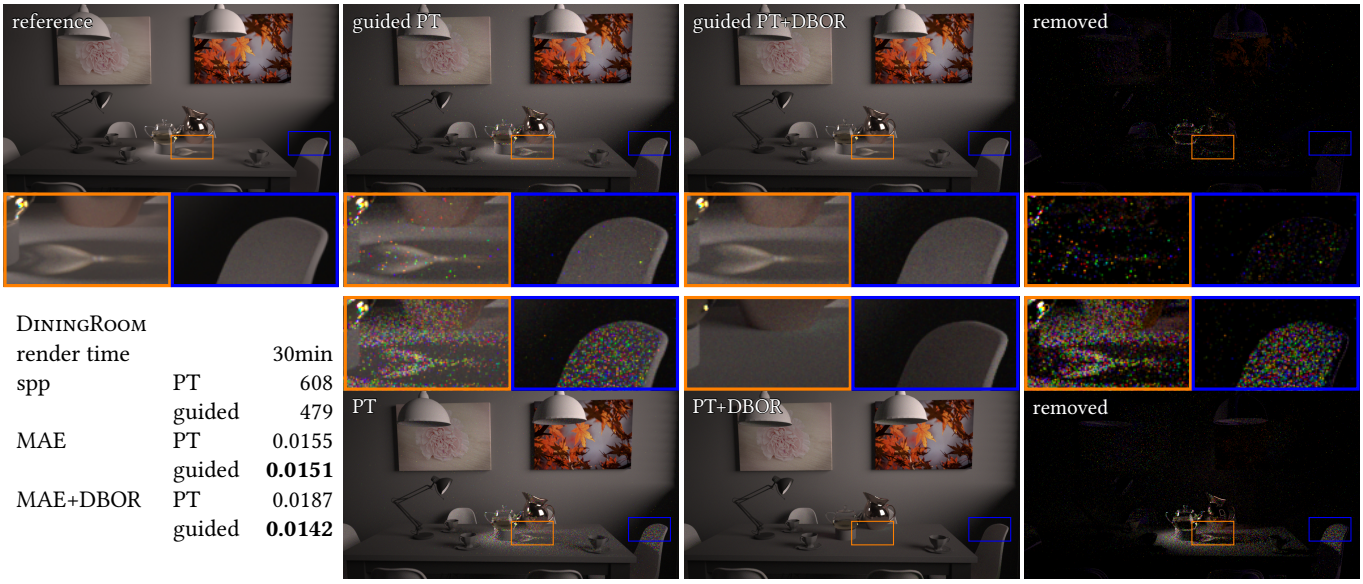
Following this train of thought, the change in half vectors $\Delta\mathbf{h}$ of a transport path can be expressed in terms of change of vertex locations $\Delta\mathbf{x}$, as a matrix [Jakob and Marschner 2012]

$$\Delta\mathbf{h} = M \cdot \Delta\mathbf{x} \tag{1}$$

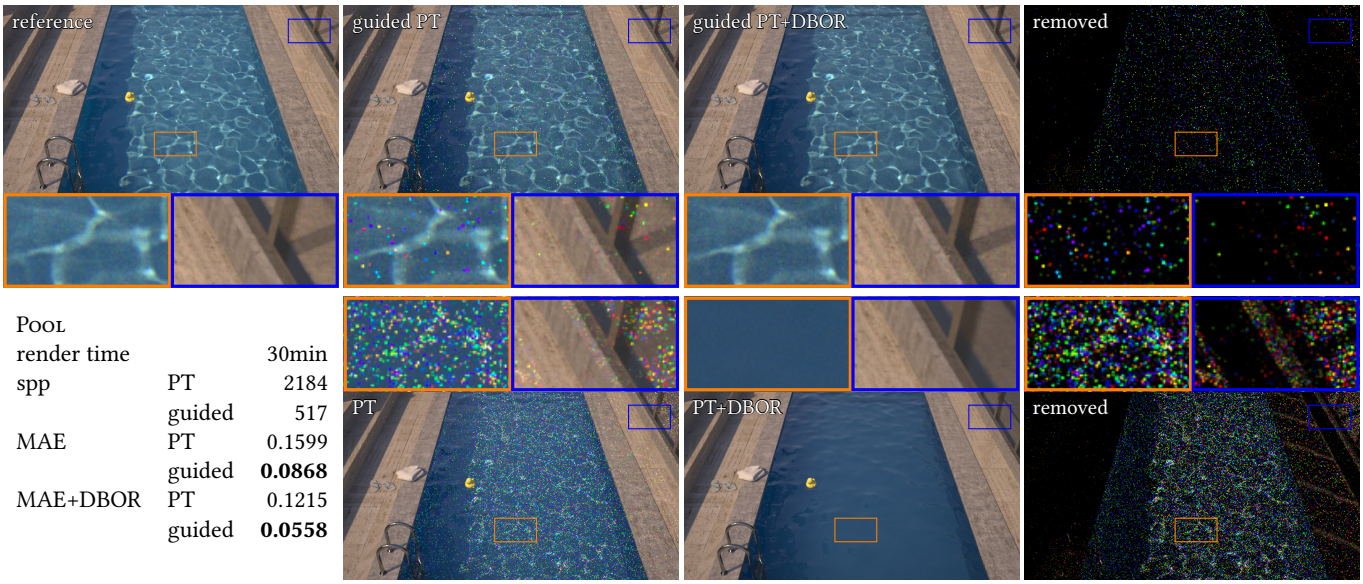
$$M = \begin{pmatrix} b_1 & c_1 & & & \\ a_2 & b_2 & c_2 & & \\ & a_3 & b_3 & c_3 & \\ & \cdots & & & \\ & & & a_k & b_k \end{pmatrix}, \tag{2}$$

where a_i, b_i, c_i are blocks of 2×2 matrices. This matrix (employed in manifold walks) formally reveals the structure of the transport operator: the throughput of the path can be separated into blocks of three consecutive vertices which fully determine the BSDF. In the general case this is also true for transmittance, geometry terms, and more general BSDF than plain microfacet models.

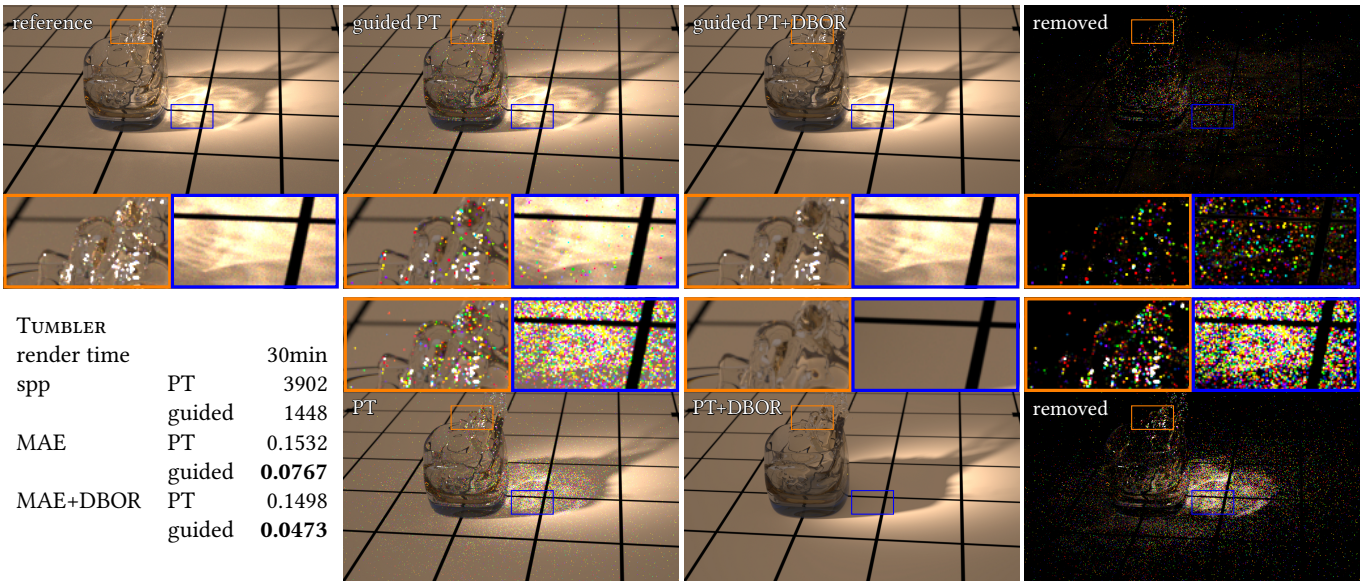
We want to find a local approximation of an ideal sampling PDF, by assuming that in a sufficiently small region around a path, the



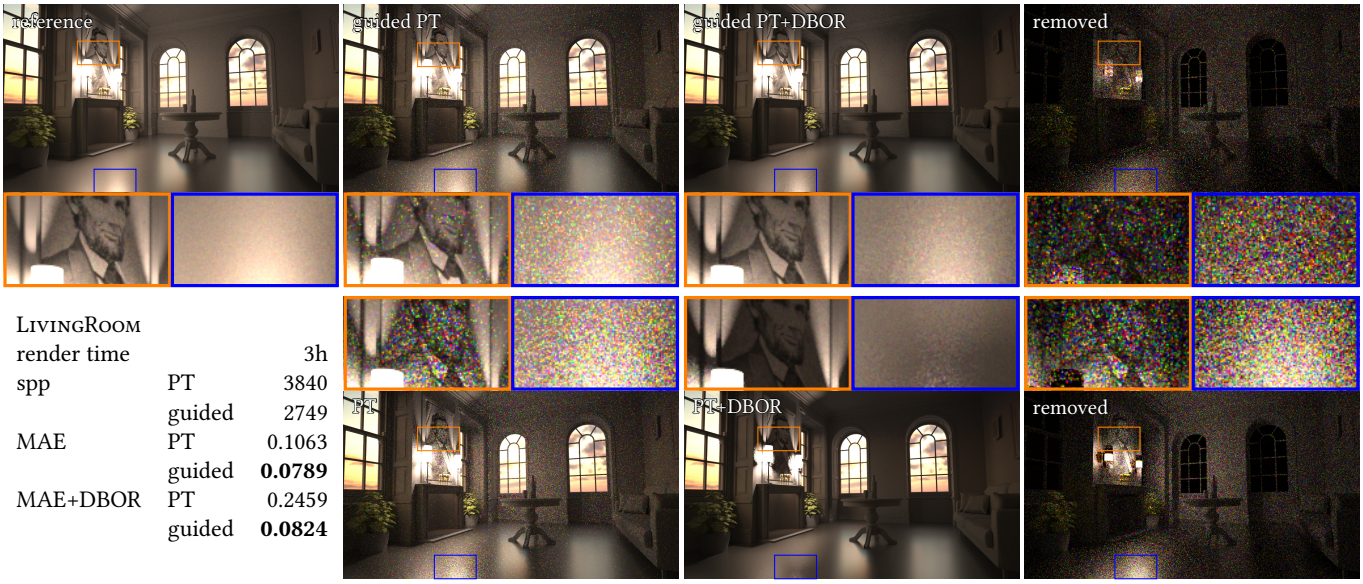
DiningRoom		
render time		30min
spp	PT	608
	guided	479
MAE	PT	0.0155
	guided	**0.0151**
MAE+DBOR	PT	0.0187
	guided	**0.0142**



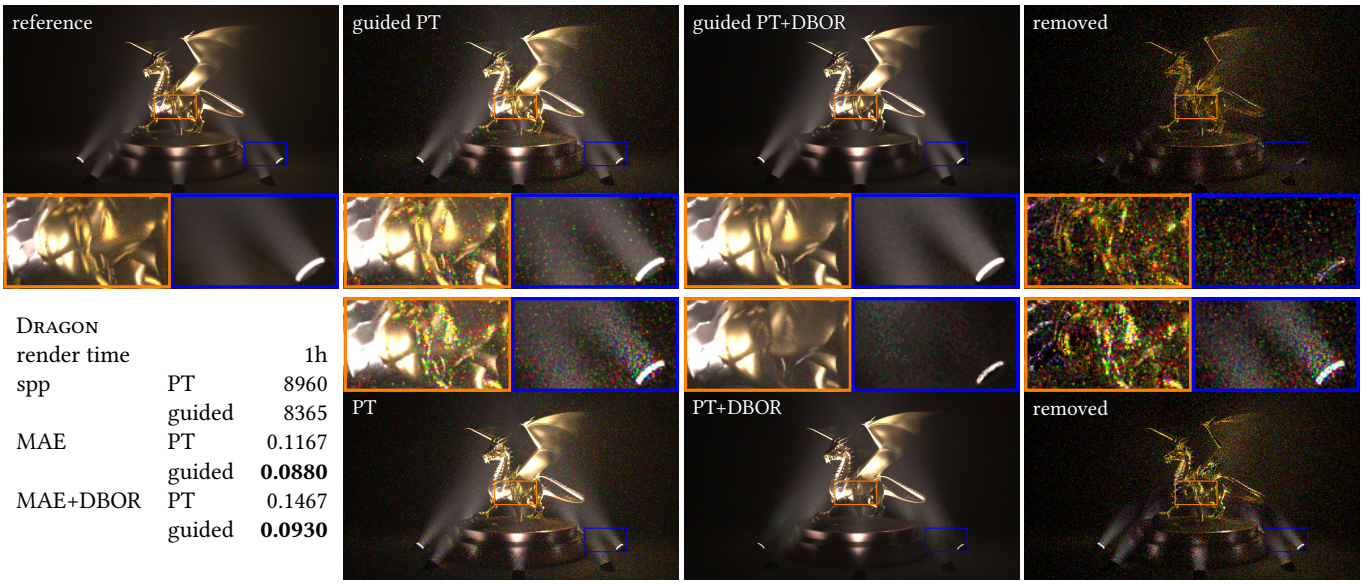
Pool		
render time		30min
spp	PT	2184
	guided	517
MAE	PT	0.1599
	guided	**0.0868**
MAE+DBOR	PT	0.1215
	guided	**0.0558**



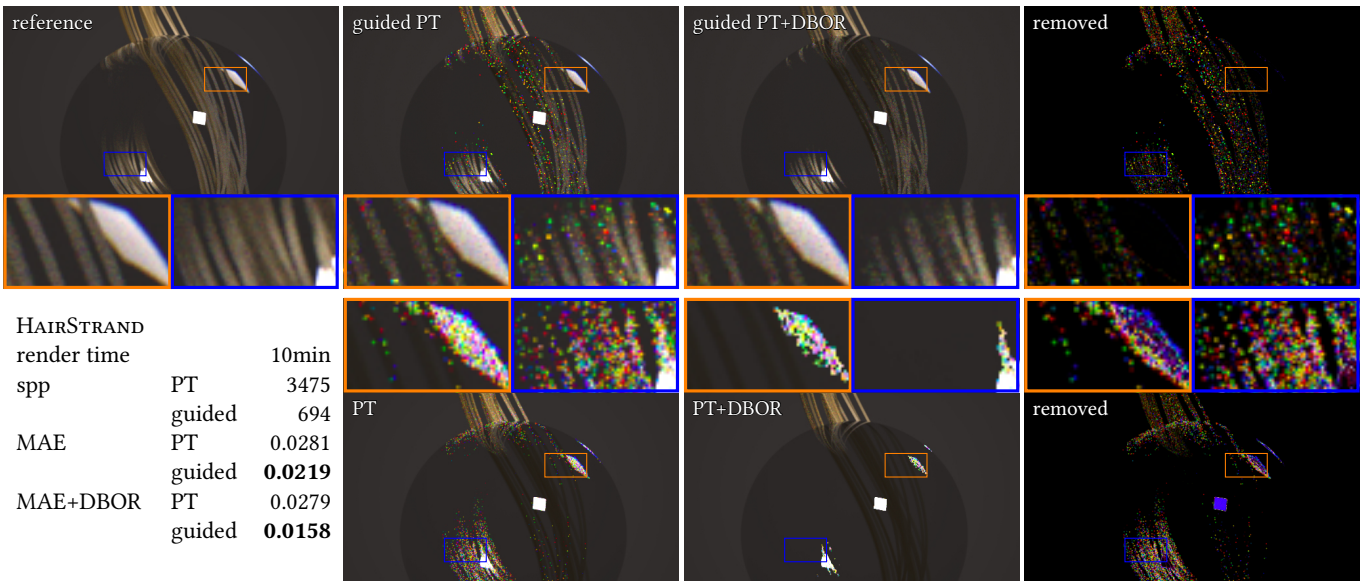
Tumbler		
render time		30min
spp	PT	3902
	guided	1448
MAE	PT	0.1532
	guided	**0.0767**
MAE+DBOR	PT	0.1498
	guided	**0.0473**



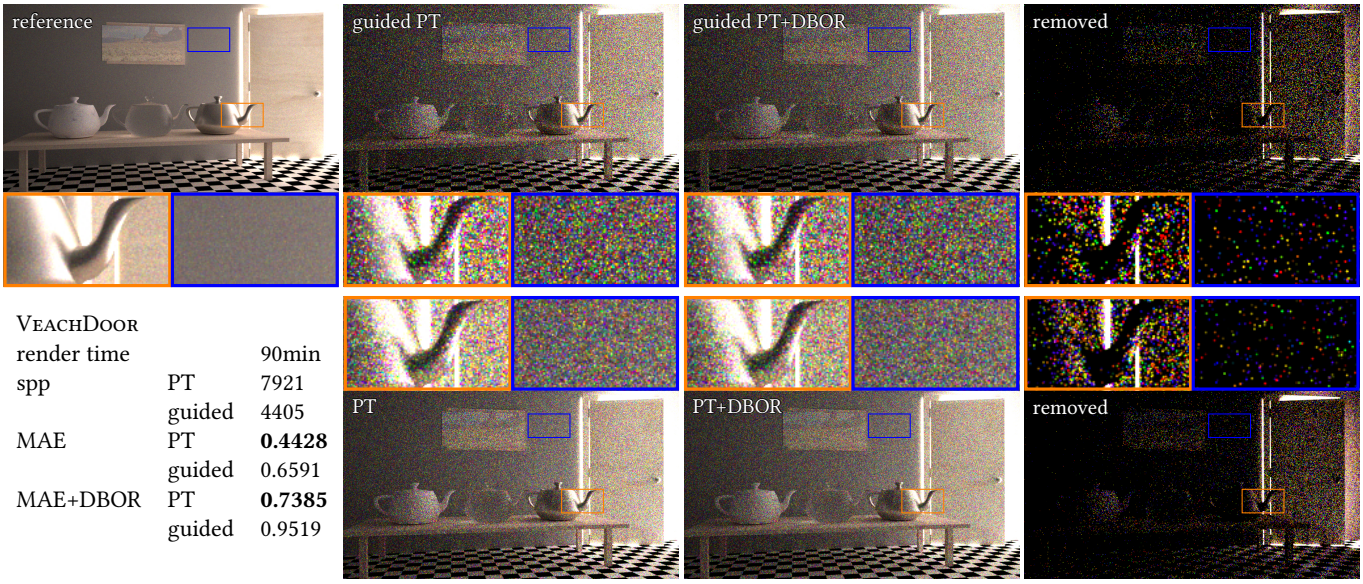
LivingRoom		
render time		3h
spp	PT	3840
	guided	2749
MAE	PT	0.1063
	guided	**0.0789**
MAE+DBOR	PT	0.2459
	guided	**0.0824**



Dragon		
render time		1h
spp	PT	8960
	guided	8365
MAE	PT	0.1167
	guided	**0.0880**
MAE+DBOR	PT	0.1467
	guided	**0.0930**



HairStrand		
render time		10min
spp	PT	3475
	guided	694
MAE	PT	0.0281
	guided	**0.0219**
MAE+DBOR	PT	0.0279
	guided	**0.0158**



VeachDoor		
render time		90min
spp	PT	7921
	guided	4405
MAE	PT	**0.4428**
	guided	0.6591
MAE+DBOR	PT	**0.7385**
	guided	0.9519

Fig. 7. Applying density-based outlier rejection (DBOR) can remove residual fireflies. Doing so removes a lot of energy when the estimator is not able to handle all features present in the scene. Our guided sampling uses a similar criterion as firefly detection to learn difficult paths, and thus running DBOR on a guided estimator removes much less energy. In fact removing outliers on the guided estimator can lead to a *lower* error. Please see Table 1 in the main paper for detailed run time, samples per pixel, and error statistics on these images. The limitations section in the main text discusses the poor performance in the VeachDoor scene.

PDF can be represented by a unimodal Gaussian. If the path has 5 path vertices which are all surface events, the resulting covariance matrix $\mathbf{S}$ of the local Gaussian distribution will be $\mathbf{S} \in \mathbb{R}^{10\times10}$. Similar to the matrix M above, each 2×2 block in $\mathbf{S}$ encodes how the distribution of vertex locations of one particular path vertex depends on the distribution of a certain other path vertex (for instance the first vertex in the scene and the vertex on the light).

Since we need to employ ray tracing to project vertices to the surface and test for visibility, we construct the path incrementally, i.e. the next vertex can only depend on the vertices sampled so far (not the others yet to be sampled). Additionally inserting the domain knowledge that the measurement contribution function is a separable product of terms which at most depend on three consecutive path vertices, we have the following structure in our covariance matrix:

$$\mathbf{S} = \begin{pmatrix} S_{11} & S_{12} & & & & \\ S_{21} & S_{22} & S_{23} & & & \\ & S_{32} & S_{33} & S_{34} & & \\ & \cdots & & & & \\ & & & S_{k(k-1)} & S_{kk} \end{pmatrix}. \tag{3}$$

As a consequence, sampling one additional vertex given a guide path and an already sampled path prefix should depend on two previous vertices. Strictly speaking we should thus form 6D covariance matrices around neighbouring guide paths and take the conditional at the current path prefix to compute a 2D covariance matrix and sample a new vertex from that (in the surface case). Since we follow the full path configuration from the start vertex, i.e. the current path prefix and the neighbouring guide paths should be very similar, we opted for a simplified variant which just takes 4D covariance matrices around two consecutive vertices and computes the conditional by only removing two dimensions instead of four. This was a big step up from not performing any conditionalisation at all (since the conditional is in general much sharper and more localised), but there may be potential in using the full 6D version instead, especially for short transport path segments.

In the setting we use in this work, we will need to compute a 4D covariance matrix for every guide path vertex. We do this with the nearest neighbours of a guide path. That is, we compute 10 distinct entries in the symmetric matrix from 10 neighbouring paths (which evaluates to 40 input dimensions, since we take two vertices with two dimensional coordinates from every path). A high ratio of input samples vs. output dimensions makes sure we arrive at useful estimates of covariance matrices.

Note that for simplicity, in the above discussion we omitted potential extra dimensions for vertices which lie in the volume. This increases both the number of entries in the covariance matrices and the input dimensions.

7 AN APPROXIMATE TRUNCATED GAUSSIAN DISTRIBUTION

Gaussian distributions are omnipresent in statistics and in Monte Carlo simulation. Often times when modelling distributions, however, one does not explicitly require it to be precisely Gaussian in shape, but only have similar behaviour with respect to the shape

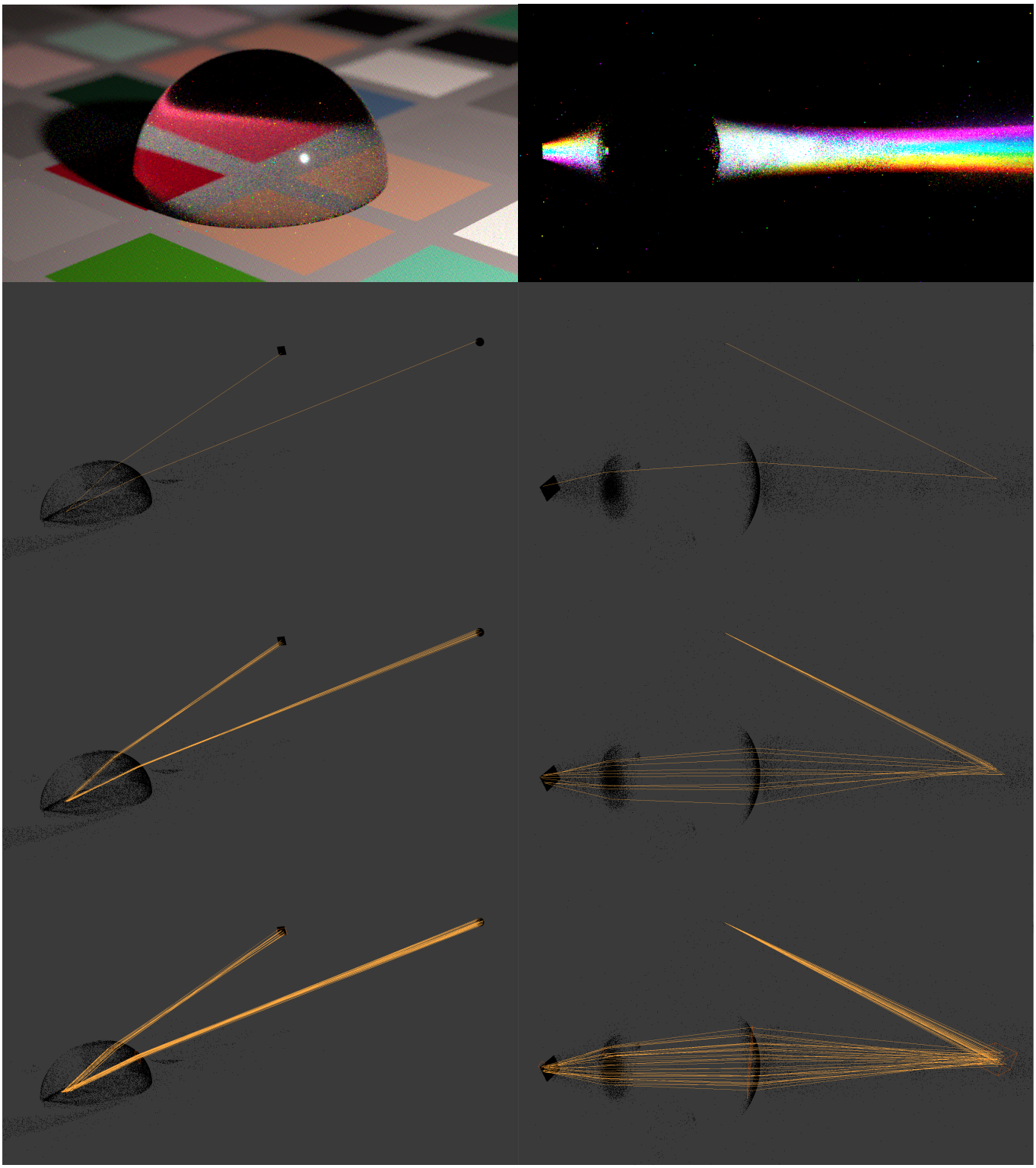


Fig. 8. A visualisation of our guided sampling in blender for the scenes from Figure 7 of the paper. The top row shows a guide path exemplar and the second row the corresponding 10 nearest guide paths in the cache. The black points show the path vertices of all guide paths in the cache. The third row shows a bounding volume (rectangle for 2D surface vertices, box in 3D for volume vertices) of the Gaussians in orange. The Gaussians were conditionalised using the guide path for visualisation purpose. The third row also shows 32 randomly sampled paths in yellow, including partial paths for which sampling fails during construction.

of the peak and tails. We examine a similar distribution that can be evaluated extremely quickly (34× speedup as compared to a Gaussian). We also devise a sampling routine to be able to use it in unbiased Monte Carlo simulations (sampling speedup is only about 2× as compared to sampling a Gaussian).

Mathematically, a Gaussian distribution has infinite support, but drops to zero in 32-bit floating point arithmetic quite quickly. We approximate a truncated Gaussian probability distribution $p(x)$ which drops to zero exactly at $p(4) = 0$, which is also taken into account by sampling.

Note that we will lose the rotational symmetry of a real multivariate Gaussian when using the approximation in a Cartesian product.

7.1 A fast approximate Gaussian PDF

Our approach is motivated by fast approximate exponentials by Paul Mineiro[1] using bit tricks:

```
fasterpow2 (float p)
{
  float clipp = (p < -126) ? -126.0f : p;
  union { uint32_t i; float f; } v =
  { (uint32_t) ( (1 << 23) * (clipp + 126.94269504f) ) };
  return v.f;
}
```

This exploits the encoding of floating point numbers by manipulating the exponent bits directly. We use this trick to compute a fast approximation of a Gaussian PDF:

```
float fake_gauss_pdf(const float x)
{
  const int k = i1 - x*x * (i2 - i1);
  return (*(const float *)&k)/norm;
}
```

Formally, the resulting PDF is a piecewise quadratic function on intervals $[\sqrt{i}, \sqrt{i+1})$:

$$p(x) = a_i \cdot x^2 + b_i, \tag{4}$$

where the coefficients of the parabola are determined for the i-th interval as

$$a_i = p\left(\sqrt{i+1}\right) - p\left(\sqrt{i}\right), \tag{5}$$

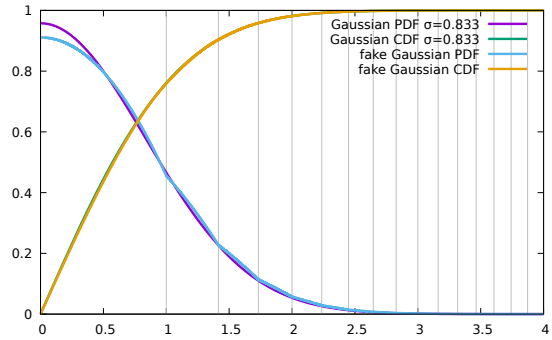
$$b_i = p\left(\sqrt{i+1}\right) - a_i \cdot (i+1), \tag{6}$$

and the PDF at the boundaries of the interval is computed and explicitly normalised as

$$p\left(\sqrt{i}\right) = 2^{-i}/n, \tag{7}$$

$$n = \int_0^\infty p'(x)\,\mathrm{d}x, \tag{8}$$

where $p'(x)$ is the not normalised PDF. For 32-bit IEEE floating point numbers, it turns out that $i \in [0, 16]$ before the result is numerically `0.0f`. The resulting intervals $\left[\sqrt{i}, \sqrt{i+1}\right)$ are indicated by the grey vertical lines in the following plot (where the PDF is not C_1 continuous), showing the PDF and CDF of a Gaussian distribution and our approximation:



Since the way the approximation works is based on powers of two instead of the natural exponential function, the PDF will approximate a Gaussian distribution with $\sigma = \sqrt{\log 2} \approx 0.833$. This means that our approximation of a standard normal distribution is truncated at $x = 4 \cdot \sqrt{\log 2} \approx 3.330$.

7.2 Sampling from the PDF

If we want to use this in a Monte Carlo simulation, we will have to sample from the approximate distribution (instead of from a real Gaussian), to ensure an unbiased estimator. The approximate Gaussian CDF can be computed analytically by performing piecewise integration of the intervals of same exponent $[\sqrt{i}, \sqrt{i+1})$. Because of the low number of intervals, we follow a pragmatic approach and precompute the 16 values of the CDF and perform a search on the resulting look up table. Since the intervals are exponentially spaced, the first one covers more than 75% of the cases. We do not want to hide that behind a search tree, and find that we can maximise performance by exploiting branch prediction which facilitates early out very efficiently for a simple linear search.

Inside each interval, we start by linear interpolation followed by up to three Newton-Raphson iterations to approximate the inverse CDF. We provide the full implementation in Section 7.3.

ACKNOWLEDGMENTS

We would like to thank Christoph Peters for a fast and robust code snippet to perform a 3×3 eigenvalue decomposition which we use for the regularisation of the truncated Gaussian kernels around our guide paths.

REFERENCES

Jiawen Chen, Sylvain Paris, and Frédo Durand. 2007. Real-time Edge-aware Image Processing with the Bilateral Grid. *ACM Trans. on Graphics (Proc. SIGGRAPH)* 26, 3 (2007).

Christopher DeCoro, Tim Weyrich, and Szymon Rusinkiewicz. 2010. Density-based Outlier Rejection in Monte Carlo Rendering. *Computer Graphics Forum (Proc. Pacific Graphics)* 29, 7 (Sept. 2010), 2119–2125.

Iliyan Georgiev, Jaroslav Křivánek, Tomáš Davidovič, and Philipp Slusallek. 2012. Light Transport Simulation with Vertex Connection and Merging. *ACM Trans. on Graphics (Proc. SIGGRAPH Asia)* 31, 6 (2012), 192:1–192:10.

Toshiya Hachisuka, Anton S. Kaplanyan, and Carsten Dachsbacher. 2014. Multiplexed Metropolis Light Transport. *ACM Trans. on Graphics (Proc. SIGGRAPH)* 33, 4, Article 100 (2014).

Johannes Hanika, Anton Kaplanyan, and Carsten Dachsbacher. 2015. Improved Half Vector Space Light Transport. *Computer Graphics Forum (Proc. Eurographics Symposium on Rendering)* 34, 4 (June 2015), 65–74.

[1] http://www.machinedlearnings.com/2011/06/fast-approximate-logarithm-exponential.html

7.3 Implementation of the approximate truncated Gaussian distribution

```
// fake gaussian distribution, normalised to (-inf,inf) or, equivalently,
// [-4,4]. it also samples this domain. close to a real gaussian with
// sigma=0.833 (i.e. sqrt(log(2))).
static inline float fakegaussian_pdf(const float x)
{
  const float norm_c1 = 2.0f * 0x1.1903a6p+0;
  const int i1 = 0x3f800000u, i2 = 0x40000000u;
  const int k0 = i1 - x*x * (i2 - i1);
  const int k = k0 > 0 ? k0 : 0;
  return (*(const float *)&k)/norm_c1;
}

static inline float fakegaussian_sample(float xi)
{
  float sign = 1.0f;
  if(xi >= 0.5f)
  {
    sign = -1.0f;
    xi = 2.0f*(xi-0.5f);
  }
  else xi *= 2.0f;

  const int i1 = 0x3f800000u, i2 = 0x40000000u;
  const float norm_c1 = 0x1.1903a6p+0;
  static const float cdf_lut[17] = {
    0x0p+0, 0x1.84aff2p-1, 0x1.ce84acp-1, 0x1.eaa068p-1,
    0x1.f66fcep-1, 0x1.fba148p-1, 0x1.fdf994p-1, 0x1.ff0d6p-1,
    0x1.ff8da8p-1, 0x1.ffc9ep-1, 0x1.ffe658p-1, 0x1.fff3ep-1,
    0x1.fffa56p-1, 0x1.fffd7p-1, 0x1.fffeeep-1, 0x1.ffffa6p-1, 0x1p+0,
  };
  static const float sqrti_lut[17] = {
    0x0p+0, 0x1p+0, 0x1.6a09e6p+0, 0x1.bb67aep+0, 0x1p+1,
    0x1.1e377ap+1, 0x1.3988e2p+1, 0x1.52a7fap+1, 0x1.6a09e6p+1,
    0x1.8p+1, 0x1.94c584p+1, 0x1.a8872ap+1, 0x1.bb67aep+1,
    0x1.cd82b4p+1, 0x1.deeea2p+1, 0x1.efbdecp+1, 0x1p+2,
  };
  static const float slope_lut[17] = {
    0x1.513794p+0, 0x1.6fad2ap+1, 0x1.7286d6p+2, 0x1.73b866p+3,
    0x1.74614p+4, 0x1.74cc88p+5, 0x1.7516c2p+6, 0x1.754d32p+7,
    0x1.7576d2p+8, 0x1.7597bp+9, 0x1.75b24ep+10, 0x1.75c84ap+11,
    0x1.75dac4p+12, 0x1.75ea8p+13, 0x1.75f814p+14, 0x1.7603e6p+15,
    0x1.760e48p+16 };
  static const float a3_lut[17] = {
    -0x1.555556p-3, -0x1.555556p-4, -0x1.555556p-5, -0x1.555556p-6,
    -0x1.555556p-7, -0x1.555556p-8, -0x1.555556p-9, -0x1.555556p-10,
    -0x1.555556p-11, -0x1.555556p-12, -0x1.555556p-13, -0x1.555556p-14,
    -0x1.555556p-15, -0x1.555556p-16, -0x1.555556p-17, -0x1.555556p-18,
    -0x1.555556p-19
  };
  static const float b_lut[17] = {
    0x1p+0, 0x1.8p-1, 0x1p-1, 0x1.4p-2,
    0x1.8p-3, 0x1.cp-4, 0x1p-4, 0x1.2p-5,
    0x1.4p-6, 0x1.6p-7, 0x1.8p-8, 0x1.ap-9,
    0x1.cp-10, 0x1.ep-11, 0x1p-11, 0x1.1p-12, 0x1.2p-13,
  };

  int i=0;
  for(;i<16;i++) if(__builtin_expect(cdf_lut[i+1] >= xi, 1)) break;

  const float cdfm = cdf_lut[i];
  const float sqrti  = sqrti_lut[i];
  const float x0 = sqrti + (xi-cdfm) * slope_lut[i];
  if(i > 3) return sign*x0; // sometimes shaves off a couple % run time, but not much.

  const float a = a3_lut[i];
  const float b = b_lut[i];
  const int k0 = i1 - x0*x0 * (i2 - i1);
  const float pdf_x0 = 1.0f/ *(const float *)&k0;
  const float bound = (cdfm - xi)*norm_c1 - (a*i + b)*sqrti;
  const float x1 = x0
    - (a*x0*x0*x0 + b*x0 + bound)*pdf_x0;
  if(i) return sign*x1;
  const float x2 = x1
    - (a*x1*x1*x1 + b*x1 + bound)*pdf_x0;
  return sign*x2;
}
```

Wenzel Jakob and Steve Marschner. 2012. Manifold Exploration: A Markov Chain Monte Carlo Technique for Rendering Scenes with Difficult Specular Transport. *ACM Trans. on Graphics (Proc. SIGGRAPH)* 31, 4 (July 2012), 58:1–58:13.

Anton Kaplanyan, Johannes Hanika, and Carsten Dachsbacher. 2014. The Natural-Constraint Representation of the Path Space for Efficient Light Transport Simulation. *ACM Trans. on Graphics (Proc. SIGGRAPH)* 33, 4 (2014), 1–13.

Csaba Kelemen, László Szirmay-Kalos, György Antal, and Ferenc Csonka. 2002. A Simple and Robust Mutation Strategy for the Metropolis Light Transport Algorithm. *Computer Graphics Forum* 21, 3 (2002), 531–540.

Eric Lafortune and Yves Willems. 1993. Bi-Directional Path Tracing. In *Proc. of COMPUGRAPHICS*. 145–153.

Eric Veach and Leonidas Guibas. 1994. Bidirectional Estimators for Light Transport. In *Proc. Eurographics Workshop on Rendering*. 147–162.

Tobias Zirr, Johannes Hanika, and Carsten Dachsbacher. 2018. Reweighting firefly samples for improved finite-sample Monte Carlo estimates. *Computer Graphics Forum* 37, 6 (2018), 410–421.