A Spatial Target Function for Metropolis Photon Tracing - Supplemental Document

1. DETAILS ON OUR FOUR-CHAIN REPLICA EXCHANGE DESIGN

In this section we provide a detailed algorithmic description of our four-chain replica exchange design for sampling from our target function. The notation used in this document is summarized in Table I. It corresponds to the main paper but we omit the iteration index i, since here we only consider a single algorithm iteration. The algorithmic steps are summarized and commented in Algorithms 1 to 5, while the main text provides further justification and details of some of these steps.

Each of the four chains uses a different target function (see Table II) and therefore distributes the paths in a different manner. Combination of all four chains via replica exchange yields a more effective algorithm than using a single chain with a complex target function. The chains with simpler target functions (uniform, visibility) help to efficiently explore the entire state space, while the remaining two chains explore local features of their complex target functions.

Algorithm 1 gives an overview of our four-chain sampling algorithm. This corresponds to the implementation of the procedure RUNMETROPOLISPATHSAMPLING from Algorithm 1 in the paper. Note that the second and the third chains generate two times fewer samples (paths) than the other chains. We found this approach to be more efficient than giving each chain the same number of samples. The algorithm uses several non-trivial procedures that are described below in separate pseudo-codes.

The procedure DOMETROPOLISUPDATE (see Algorithm 2) is used for mutations of a single chain according to the Metropolis-Hastings algorithm [Hastings 1970], while DOREPLICAEXCHANGE (see Algorithm 3) handles replica exchange between two neighboring chains. For each chain, we use an independent adaptive MCMC process to automatically tune the mutation size [Hachisuka and Jensen 2011].

Please note that the first two chains (uniform, visibility) have such simple target functions that their mutations and replica exchange between them can be easily simplified. We omit this simplification in the pseudocode presented here but we do use it in the implementation itself. Please refer to the work of Hachisuka and Jensen [2011] for details.

The procedure UPDATENORMALIZATION (see Algorithm 4) estimates the normalization constant b_{L_c} of one chain (L_c) using the samples from the previous chain (L_{c-1}). The procedure actually

L_c	c-th chain
$X_{L_c}^{(j)}$	j-th state of the c-th chain (corresponds to a single light sub-path)
Y	proposed state (light sub-path)
$\hat{T}_{L_c}$	target function of the c-th chain
b_{L_c}	normalization constant of $\hat{T}_{L_c}$, $b_{L_c} = \int \hat{T}_{L_c}(\bar{\mathbf{u}})\mathrm{d}\bar{\mathbf{u}}$
$b_{L_c}^{(p)}$	normalization constant of $\hat{T}_{L_c}$ from a previous iteration
$update_{L_1}$	variable used for alternating the updates of chains L_1 and L_2
$accum_{L_c}$	sample accumulator used for the computation of the normalization b_{L_c}
M	number of cycles of the main sampling loop in each iteration
$samples_{L_c}$	number of samples generated by the c-th chain in a single iteration (M for L_0 and L_3; $M/2$ for L_1 and L_2)
G_k	measurement point
$P[G_k]$	RGB value of the pixel associated with a measurement point G_k
$\mathcal{U}$	the primary-sample space
$\bar{\mathbf{u}}$	a variable from $\mathcal{U}$
$V(X)$	a visibility function that returns 1 for paths that contribute to at least one measurement point and 0 to non-contributing paths
$S(X)$	an inverse size of a measurement point kernel (the right factor in Eq. 11 in the paper)
$D(X)$	an inverse of a density D^{uni} (the left factor in Eq. 11 in the paper)

Table I. : Notation used throughout this supplemental document.

L_0 (Uniform):	$\hat{T}_{L_0}(X) = 1$
L_1 (Visibility):	$\hat{T}_{L_1}(X) = V(X)$
L_2 (Inverse kernel size):	$\hat{T}_{L_2}(X) = V(X)S(X)$
L_3 (Our target function):	$\hat{T}_{L_3}(X) = V(X)S(X)D(X)$

Table II. : The four different chains and their target functions.

Algorithm 1 Pseudo-code of the 4-chain algorithm.

for $j = 1 \dots M$ **do**
  // Advance the uniform (L_0) chain
  $X_{L_0}^{(j)} = \text{DOMETROPOLISUPDATE}(L_0 : X_{L_0}^{(j-1)})$
  // Update the normalization of the visibility (L_1) chain
  $\text{UPDATENORMALIZATION}(b_{L_1} : X_{L_0}^{(j)})$
  // Perform replica exchange between L_0 and L_1
  $\text{DOREPLICAEXCHANGE}(L_0 : X_{L_0}^{(j)},\ L_1 : X_{L_1}^{(j-1)})$
  // Alternate the update of chains L_1 and L_2
  if $update_{L_1} == true$ **then**
    $c = 1, X_{L_2}^{(j)} = X_{L_2}^{(j-1)}$
  else
    $c = 2, X_{L_1}^{(j)} = X_{L_1}^{(j-1)}$
  end if
  $update_{L_1} = \neg update_{L_1}$ // Alternate
  // Advance the selected L_c chain
  $X_{L_c}^{(j)} = \text{DOMETROPOLISUPDATE}(L_c : X_{L_c}^{(j-1)})$
  // Update the normalization of the next (L_{c+1}) chain
  $\text{UPDATENORMALIZATION}(b_{L_{c+1}} : X_{L_c}^{(j)})$
  // Perform replica exchange between L_c and L_{c+1}
  $\text{DOREPLICAEXCHANGE}(L_c : X_{L_c}^{(j)},\ L_{c+1} : X_{L_{c+1}}^{(j-1)})$
  // Finally, advance the L_3 chain (our TF)
  $X_{L_3}^{(j)} = \text{DOMETROPOLISUPDATE}(L_3 : X_{L_3}^{(j-1)})$
end for

Algorithm 2 DoMetropolisUpdate pseudo-code.

function DoMetropolisUpdate($L_c : X$)
  // Generate a proposal sample
  $Y =$ Mutate($L_c : X$)
  // Calculate acceptance probability
  $a = \min\{1, \frac{\hat{T}_{L_c}(Y)}{\hat{T}_{L_c}(X)}\}$
  // Splat the expected value of the current and proposed state
  Splat($L_c : Y, a$)
  Splat($L_c : X, 1 - a$)
  // Accept/Reject the proposal
  if $a >$Random() **then**
    return Y
  else
    return X
  end if
end function

Algorithm 3 DoReplicaExchange pseudo-code.

function DoReplicaExchange($L_c : X_{L_c},\ L_{c+1} : X_{L_{c+1}}$)
  // Compute the swap probability for chains L_c and L_{c+1}
  $r = \min\{1, \frac{\hat{T}_{L_c}(X_{L_{c+1}})}{\hat{T}_{L_c}(X_{L_c})} \frac{\hat{T}_{L_{c+1}}(X_{L_c})}{\hat{T}_{L_{c+1}}(X_{L_{c+1}})}\}$
  // Accept/reject the swap
  if $r >$Random() **then**
    SwapStates($X_{L_c}, X_{L_{c+1}}$)
  end if
end function

Algorithm 4 UpdateNormalization pseudo-code.

function UpdateNormalization($b_{L_{c+1}} : X_{L_c}$)
  // Accumulate the current sample
  $accum_{L_{c+1}} += \frac{\hat{T}_{L_{c+1}}(X_{L_c})}{\hat{T}_{L_c}(X_{L_c})}$
  // Compute the updated normalization
  $b_{L_{c+1}} = b_{L_c} \frac{accum_{L_{c+1}}}{samples_{L_c}}$
end function

implements a Monte Carlo estimation of the following formula

$$b_{L_c} = b_{L_{c-1}} \int_{\mathcal{U}} \frac{\hat{T}_{L_c}(\bar{\mathbf{u}})}{\hat{T}_{L_{c-1}}(\bar{\mathbf{u}})} d\bar{\mathbf{u}}. \quad (1)$$

Note that the formula requires $b_{L_{c-1}}$ (the normalization of the previous chain) to be computed beforehand. This leads to a recursive computation starting with $b_{L_0} = 1$.

Finally, the procedure Splat (see Algorithm 5) is responsible for evaluating the contribution of the photons from one sample (each sample corresponds to a single light path) to the nearest measurement points. The accumulated pixel value $P[G_k]$ must be normalized after each iteration, which is achieved by multiplying it with

$$\frac{b_{L_c}}{samples_{L_c}}. \quad (2)$$

Multiple importance sampling (MIS) between the chains is also evaluated in Splat. The MIS weight calculation requires to evaluate the pdf of all sampled paths for the four sampling techniques corresponding to the four chains. These pdfs are given by the respective target functions divided by their normalization constants, which themselves are only determined at the very end of the iteration. For this reason, we take an approximation in calculating the MIS weights, and use the normalization constants ($b_{L_c}^{(p)}$) from the previous iteration. Note that these approximate MIS weights do not introduce any additional bias because they still sum up to one.

Algorithm 5 Splat pseudo-code.

function Splat($L_c : X_{L_c}, weight$)
  // Find measurement points around the photons of X_{L_c}
  $\mathcal{G} =$ FindMeasurementPoints(X_{L_c})
  // Compute MIS weight of the sample
  $weight_{\text{MIS}} = \frac{\left(samples_{L_c} \frac{\hat{T}_{L_c}(X_{L_c})}{b_{L_c}^{(p)}}\right)^2}{\sum_{i=0}^{3}\left(samples_{L_i} \frac{\hat{T}_{L_i}(X_{L_c})}{b_{L_i}^{(p)}}\right)^2}$
  for each measurement point $G_k \in \mathcal{G}$ **do**
    // Compute the radiance from a photon of X_{L_c}
    // to a measurement point G_k
    $radiance =$ GetRadiance(G_k, X_{L_c})
    // Splat the value to a pixel associated with G_k
    $P[G_k] += (radiance)(weight_{\text{MIS}}) \frac{weight}{\hat{T}_{L_c}(X_{L_c})}$
  end for
end function

REFERENCES

Hachisuka, T. and Jensen, H. W. 2011. Robust adaptive photon tracing using photon path visibility. *ACM Trans. Graph. 30,* 5 (Oct.).

Hastings, W. K. 1970. Monte carlo sampling methods using markov chains and their applications. *Biometrika 57,* 1, pp. 97–109.

Kitaoka, S., Kitamura, Y., and Kishino, F. 2009. Replica exchange light transport. *Computer Graphics Forum 28,* 8, 2330–2342.