Fast Volume Rendering with Spatiotemporal Reservoir Resampling (Supplemental Document)

DAQI LIN, University of Utah, USA
CHRIS WYMAN, NVIDIA, USA
CEM YUKSEL, University of Utah, USA

ACM Reference Format:
Daqi Lin, Chris Wyman, and Cem Yuksel. 2021. Fast Volume Rendering with Spatiotemporal Reservoir Resampling (Supplemental Document). *ACM Trans. Graph.* 40, 6, Article 279 (December 2021), 4 pages. https://doi.org/10.1145/3478513.3480499

1 RIS FOR PATH INTEGRALS

This subsection provides a more rigorous derivation of the RIS estimator of the path integral proposed in Section 3.3 in the paper. We assume the path tracing process being a random walk that generates n paths $\boldsymbol{\lambda}^i (i \in \{1, ..., n\})$ that differ by lengths (or emission/scatter type in the volumetric case), responsible for n non-overlapping parts $(\int_{\Lambda^i} F(\boldsymbol{\lambda}^i)\, d\boldsymbol{\lambda}^i)$ of the path integral $L = \int_{\Lambda} F(\boldsymbol{\lambda})\, d\boldsymbol{\lambda}$ (where $\boldsymbol{\lambda}$ can be any length or type). In other words, $L = \sum_{i=1}^{\infty} \int_{\Lambda^i} F(\boldsymbol{\lambda}^i)\, d\boldsymbol{\lambda}^i$. An ordinary 1-sample Monte Carlo estimator simply sums the contributions from all sampled paths, i.e. $\langle L \rangle_{\text{MC}} = \sum_{i=1}^{n} \frac{F(\boldsymbol{\lambda}^i)}{p(\boldsymbol{\lambda}^i)}$. Note that n is a random variable and can pick any value from 0 to ∞. We know that $\langle L \rangle_{\text{MC}}$ is an unbiased estimator of the path integral L. However, we want to take advantage of RIS to avoid evaluating F for all paths. The traditional form of RIS assumes that all candidate samples are taken from the same sampling domain to estimate the same integral. Now, we want to use RIS to estimate a sum of integrals using candidate samples taken from the sampling domain of each integral, and only evaluate F for the chosen sample $\boldsymbol{\lambda}^r$. This requires a slightly different definition of RIS (the "$1/M$" term is removed from the estimator). We now show that the RIS estimator

$$\langle L \rangle_{\text{ris}} = \frac{F(\boldsymbol{\lambda}^r)}{\hat{p}(\boldsymbol{\lambda}^r)} \sum_{i=1}^{n} \frac{\hat{p}(\boldsymbol{\lambda}^i)}{p(\boldsymbol{\lambda}^i)}$$

where $\boldsymbol{\lambda}^r$ is chosen from $\boldsymbol{\lambda}^1, ..., \boldsymbol{\lambda}^n$ according to the weight $w(\boldsymbol{\lambda}^i) = \frac{\hat{p}(\boldsymbol{\lambda}^i)}{p(\boldsymbol{\lambda}^i)}$ is also an unbiased estimator of the path integral, i.e. $\mathbb{E}[\langle L \rangle_{\text{ris}}] = \mathbb{E}[\langle L \rangle_{\text{MC}}]$. The proof goes as follows,

$$\mathbb{E}\left[\langle L \rangle_{\text{ris}}\right] = \mathbb{E}\left[\mathbb{E}\left[\frac{F(\boldsymbol{\lambda}^r)}{\hat{p}(\boldsymbol{\lambda}^r)} | \boldsymbol{\lambda}^1, ..., \boldsymbol{\lambda}^n\right] \cdot \sum_{i=1}^{n} w(\boldsymbol{\lambda}^i)\right]$$

$$= \mathbb{E}\left[\sum_{j=1}^{n} \left(\frac{F(\boldsymbol{\lambda}^j)}{\hat{p}(\boldsymbol{\lambda}^j)} \cdot \frac{w(\boldsymbol{\lambda}^j)}{\sum_{i=1}^{n} w(\boldsymbol{\lambda}^i)}\right) \cdot \sum_{i=1}^{n} w(\boldsymbol{\lambda}^i)\right]$$

$$= \mathbb{E}\left[\sum_{j=1}^{n} \frac{F(\boldsymbol{\lambda}^j)}{\hat{p}(\boldsymbol{\lambda}^j)} \cdot \frac{\hat{p}(\boldsymbol{\lambda}^j)}{p(\boldsymbol{\lambda}^j)}\right]$$

$$= \mathbb{E}\left[\sum_{j=1}^{n} \frac{F(\boldsymbol{\lambda}^j)}{p(\boldsymbol{\lambda}^j)}\right] = \mathbb{E}[\langle I \rangle_{MC}]. \quad \square$$

2 COMPARISON WITH VERTEX REUSE

Section 4.2 in the paper notes we can either resample paths by reusing path vertices $\mathbf{x}_i$ or by reusing directions $\boldsymbol{\omega}_i$. We chose to reuse directions; while costs are somewhat higher for direction reuse, it reduces noise fairly significantly. This is because paths act a little like virtual point lights under reuse, which leads to more singularities and fireflies. However, if these could be reduced, it may make sense to switch back to vertex reuse for the improved performance. See Figure 1 to compare visually between vertex and directional reuse.

3 DENOISED RESULT

In Figure 2, we compare denoising applied to both our baseline and our resampling technique; both using the new OptiX 7.3 temporal denoising mode. While OptiX produces amazingly denoised results in both cases, the better sampling provided by our technique preserves much higher frequency details in the animation, while the baseline gives a smoother, more washed out look. Part of this is also due to the improved motion vectors we provide with our novel temporal reprojection plus velocity resampling.

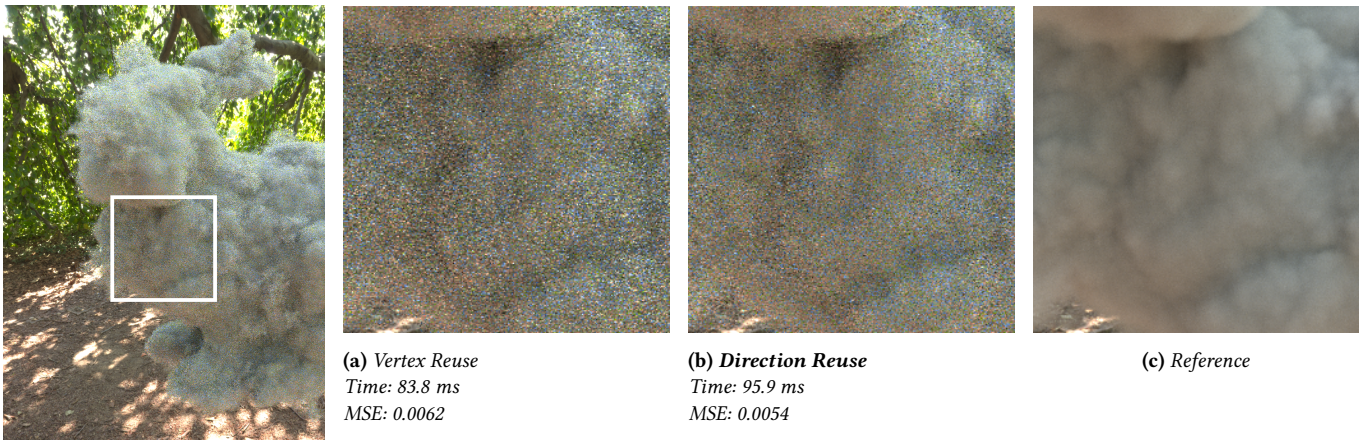
Please see the supplementary video to compare the denoised results under animation.

4 PSEUDOCODE

We provide the pseudocode of our fast volume rendering with spatiotemporal reservoir resampling (Volumetric ReSTIR) in this supplemental document. Algorithm 1 gives an outline of the full algorithm, ignoring surface bounces for simplicity. Algorithm 2 shows the reservoir class we used for storing and streaming the samples. Algorithm 3, Algorithm 5, and Algorithm 6 shows four individual stages of our algorithm: initial sampling, temporal reuse, spatial reuse, and final shading. We provide the detail of how we resample the target function $\hat{p}$ in Algorithm 4.

Authors' addresses: Daqi Lin, University of Utah, USA; Chris Wyman, NVIDIA, USA; Cem Yuksel, University of Utah, USA.

0730-0301/2021/12-ART279 $15.00
https://doi.org/10.1145/3478513.3480499



(a) *Vertex Reuse*
Time: 83.8 ms
MSE: 0.0062

(b) ***Direction Reuse***
Time: 95.9 ms
MSE: 0.0054

(c) *Reference*

Fig. 1. *Compare vertex and direction reuse. (We reuse directions.) Here we show 7-bounce multiple scattering in the Bunny Cloud scene. While vertex reuse provides a slight performance win, it introduces additional sampling variance due to large weights of potentially irrelevant paths during spatial reuse.*

Baseline (1 bounce) with denoising

Ours *(1 bounce) with denoising*

Baseline (7 bounces) with denoising

Ours *(7 bounces) with denoising*

Fig. 2. *The OptiX 7.3 temporal denoiser applied to both our baseline and our method, using (top) single and (bottom) 7-bounce multiple scattering.*

Algorithm 1: Full algorithm.

```
function VolumetricReSTIR()
    allocate an image size array of reservoirs
    foreach q ∈ Image do
        reservoirs[q] ← InitialSampling(q)
    end
    foreach q ∈ Image do
        reservoirs[q] ← TemporalReuse(q)
    end
    foreach q ∈ Image do
        reservoirs[q] ← SpatialReuse(q)
    end
    prevFrameReservoirs ← reservoirs
    foreach q ∈ Image do
        pixel color ← FinalShading(q)
    end
```

Algorithm 2: Pseudocode of the Reservoir class.

```
class Reservoir
    λ ← null
    // the content of the path sample can be seen in line 3, Algorithm 4.
    w_sum ← 0
    M ← 0
    p̂ ← 0
    function update(p_i, p̂_i, λ_i)
        if null sample then
            M ← M + 1
        else
            w_i ← p̂_i / p_i
            w_sum ← w_sum + w_i
            M ← M + 1
            // Generate a random number ξ.
            if ξ < w_i / w_sum then
                λ ← λ_i
                p̂ ← p̂_i
    function combineReservoir(p̂_i, r_i, m)
        w_i ← p̂_i / (r_i.p̂) · r_i.w_sum · m
        w_sum ← w_sum + w_i
        M ← M + r_i.M
        if ξ < w_i / w_sum then
            λ ← λ_i
            p̂ ← p̂_i
```

Algorithm 3: Pseudocode of the initial sampling pass.

```
function InitialSampling(q)
    Reservoir r
    [x_0, ω_0] ← ray origin and direction from pixel q
    // M is the number of random walks
    for m ← 1 to M do
        Reservoir r_m
        p̂ ← 1
        p ← 1
        λ_0 ← Path with only camera vertex
        for i ← 1 to K do
            [z_i, σ*_t(x_i), T*] ← RegularTracking(x_{i−1}, ω_{i−1})
            λ_i ← AddPathVertex(λ_{i−1}, x_i)
            if z_i == z_s then
                if i == 1 then
                    p ← p · T*
                    p̂ ← p̂ · L^s T*
                    r_m.update(p, p̂, λ_i)
                else
                    r_m.update(null sample)
                break
            if σ_t(x_i) == 0 then
                r_m.update(null sample)
                break
            p ← p · σ*_t(x_i) T*
            p̂ ← p̂ · T*
            if volume contains emission then
                p̂′ ← p̂ · σ_a(x_i) L^m_e(x_i)
                // the superscript e marks an emission path
                r_m.update(p, p̂′, λ^e_i)
            // Scattering event. (p_NEE is the pdf of the light sample)
            [L_s, x′_{i+1}, ω′_i, p_NEE] ← SampleLight (x_i)
            // ray marching with a larger step size
            T̃ ← EstimateVolumetricShadow(x_i, x′_{i+1})
            p̂ ← p̂ · σ_s(x_i) ρ(x_i, −ω_{k−1}, ω′_i) L_s T̃
            p ← p · p_NEE
            r_m.update(p, p̂, AddPathVertex(λ_i, x′_{i+1}))
            // Prepare for the next bounce.
            ω_i ← SamplePhaseFunction(x_i)
            p̂ ← p̂ · σ_s(x_i) ρ(x_i, −ω_{i−1}, ω_i)
            p ← p · ρ(x_i, −ω_{i−1}, ω_i)
        end
        // Set M to 1 since we are using the path integral RIS.
        r_m.M ← 1
        r.combineReservoir(p̂(r_m.λ), r_m, 1)
    end
    return r
```

Algorithm 4: Resample target function.

function `ResampleTargetFunction`($\boldsymbol{\lambda}, q$)
  $[\mathbf{x}_0, \boldsymbol{\omega}_0] \leftarrow$ ray origin and direction from pixel q
  // k' is the number of scattering events (plus 1 for emissive paths [1]). $\mathbf{x}_{k'+1}$ and $\boldsymbol{\omega}_{k'}$ are the position and direction of the light sample.
  $[k', isEmission, z_1, \boldsymbol{\omega}_1, z_2, \boldsymbol{\omega}_2, ..., z_{k'}, \boldsymbol{\omega}_{k'}, \mathbf{x}_{k'+1}] \leftarrow \boldsymbol{\lambda}$
  $\hat{p} \leftarrow 1$
  for $i \leftarrow 1$ **to** k' **do**
    $\mathbf{x}_i = \mathbf{x}_{i-1} + z_i \boldsymbol{\omega}_{i-1}$ // Direction reuse
    $\tilde{T} \leftarrow \text{RayMarching}(\mathbf{x}_{i-1}, \mathbf{x}_i)$
    if $z_i == z_s$ **then**
      if $i == 1$ **then**
        $\hat{p} \leftarrow L^s \tilde{T}$
        return $\hat{p}$
      else
        return 0
    if *isEmission* **then**
      return $\hat{p} \cdot \sigma_a(\mathbf{x}_i) \tilde{T} L_e^m(\mathbf{x}_i)$
    $\hat{p} \leftarrow \hat{p} \cdot \sigma_s(\mathbf{x}_i) \tilde{T}$
    $\hat{p} \leftarrow \hat{p} \cdot \rho(\mathbf{x}_i, -\boldsymbol{\omega}_{i-1}, \boldsymbol{\omega}_i)$
  end
  // Transmittance of the NEE segment
  $\tilde{T} \leftarrow \text{RayMarching}(\mathbf{x}_{k'}, \mathbf{x}_{k'+1})$
  $\hat{p} \leftarrow \hat{p} \cdot \tilde{T}$
  return $\hat{p}$

Algorithm 5: Pseudocode of spatial/temporal reuse.

function `TemporalReuse`(q)
  $q' \leftarrow \text{TemporalReprojection}(q, \text{reservoirs}[q])$
  return CombineReservoirs(reservoirs[q], prevFrameReservoirs[q'], q, q'})
function `SpatialReuse`(q)
  $S \leftarrow \text{pickSpatialNeighbors}(q)$
  return CombineReservoirs(reservoirs[q], {reservoir[q'] | $q' \in S$ }, q, {$q' \in S$})
function `CombineReservoirs`($r_0, r_1, ..., r_N, q_0, q_1, ..., q_N$)
  Reservoir s
  foreach $r_i \in \{r_0, ..., r_N\}$ **do**
    $p_{\text{sum}} \leftarrow 0$
    $k \leftarrow 0$
    // Compute MIS weight
    foreach $q_s \in \{q_0, ..., q_N\}$ **do**
      $\hat{p}_{q_s}(r_i.\boldsymbol{\lambda}) \leftarrow \text{ResampleTargetFunction}(r_i.\boldsymbol{\lambda}, q_s)$
      $p_{\text{sum}} \leftarrow p_{\text{sum}} + \hat{p}_{q_s}(r_i.\boldsymbol{\lambda}) \cdot r_i.M$
      $k \leftarrow k + r_i.M$
    end
    $m \leftarrow \frac{\hat{p}_{q_i}(r_i.\boldsymbol{\lambda})}{p_{\text{sum}}/k}$ // MIS weight
    $\hat{p}_q(r_i.\boldsymbol{\lambda}) \leftarrow \text{ResampleTargetFunction}(r_i.\boldsymbol{\lambda}, q_0)$
    s.combineReservoir($\hat{p}_q(r_i.\boldsymbol{\lambda})$, r_i, m)
  end
  return s

Algorithm 6: Final shading.

function `FinalShading`(q)
  $r \leftarrow \text{reservoirs}[q]$
  // Analytical transmittance values are computed.
  $f(r.\boldsymbol{\lambda}) \leftarrow$ compute integrand
  return $\frac{f(r.\boldsymbol{\lambda})}{\hat{p}(r.\boldsymbol{\lambda})} \frac{r.w_{sum}}{r.M}$

[1] Directly visible surface emission also counts as an emission path, with the corresponding terms ("if emission path") in Equation 15 and 25 in the paper changed to $L^s(\mathbf{x}_1 \rightarrow \mathbf{x}_0)$.