Path Replay Backpropagation: Differentiating Light Paths using Constant Memory and Linear Time — Supplemental Material

DELIO VICINI, École Polytechnique Fédérale de Lausanne (EPFL), Switzerland
SÉBASTIEN SPEIERER, École Polytechnique Fédérale de Lausanne (EPFL), Switzerland
WENZEL JAKOB, École Polytechnique Fédérale de Lausanne (EPFL), Switzerland

1 ATTACHED DERIVATIVES: RECURSIVE FORMULATION

Equation (5) in the main paper specifies an energy balance equation for differential light transport in the *detached* case and is obtained by differentiating the rendering equation. Here, we perform a similar derivation for the case of attached derivatives, starting with the rendering equation (without emission) expressed as an integral over primary sample space:

$$\begin{aligned} L_o(\mathbf{x}, \boldsymbol{\omega}) &= \int_{S^2} L_i(\mathbf{x}, \boldsymbol{\omega}') f_s(\mathbf{x}, \boldsymbol{\omega}, \boldsymbol{\omega}') \, \mathrm{d}\boldsymbol{\omega}'^{\perp} \\ &= \int_{\mathcal{U}^2} L_i(\mathbf{x}, T(\mathbf{u}, \pi)) \frac{f_s(\mathbf{x}, \boldsymbol{\omega}, T(\mathbf{u}, \pi))}{p(\mathbf{x}, \boldsymbol{\omega}, T(\mathbf{u}, \pi))} \, \mathrm{d}\mathbf{u}, \end{aligned} \tag{1}$$

where T maps uniformly distributed random variates $\mathbf{u} \in [0, 1)^2$ to sampled directions, and p is the solid angle density of the generated samples. Similar to the derivation for detached derivatives, we can now compute the derivative with respect to the scene parameter π:

$$\begin{aligned} \partial_\pi L_o(\mathbf{x}, \boldsymbol{\omega}) = \int_{\mathcal{U}^2} & L_i(\mathbf{x}, T(\mathbf{u}, \pi)) \; \partial_\pi \left[\frac{f_s(\mathbf{x}, \boldsymbol{\omega}, T(\mathbf{u}, \pi))}{p(\mathbf{x}, \boldsymbol{\omega}, T(\mathbf{u}, \pi))} \right] \\ & + \partial_\pi L_i(\mathbf{x}, T(u, \pi)) \; \frac{f_s(\mathbf{x}, \boldsymbol{\omega}, T(\mathbf{u}, \pi))}{p(\mathbf{x}, \boldsymbol{\omega}, T(\mathbf{u}, \pi))} \\ & + \partial_{\omega'} L_i(\mathbf{x}, T(\mathbf{u}, \pi)) \; \cdot \partial_\pi \left[T(\mathbf{u}, \pi) \right] \; \frac{f_s(\mathbf{x}, \boldsymbol{\omega}, T(\mathbf{u}, \pi))}{p(\mathbf{x}, \boldsymbol{\omega}, T(\mathbf{u}, \pi))} \, \mathrm{d}\mathbf{u} \end{aligned} \tag{2}$$

We used the product rule and the multivariate chain rule to split the derivative into three separate terms. The first two terms involve the primal incident radiance and its parametric derivative. They resemble the detached case and can therefore be evaluated analogously. However, the third term is new and requires the *directional derivative* of the incident radiance. To obtain this new type of derivative at each bounce of the adjoint pass, we initially compute the derivative of the incident radiance with respect to the camera ray and then perform appropriate transformations (Listing 3) while re-tracing the path during the adjoint pass.

2 UNBIASEDNESS OF PROBABILISTIC REGULARIZATION

As discussed above, the adjoint phase of our attached estimator requires an unbiased estimate of the directional derivative of the incident radiance at every path vertex. Regularization (discussed in the main paper) is necessary to avoid situations in which the associated computation becomes ill-defined. We now justify that this regularization of the Jacobian and the associated inversion of noisy matrices does not introduce bias.

Authors' addresses: Delio Vicini, delio.vicini@epfl.ch, École Polytechnique Fédérale de Lausanne (EPFL), Lausanne, Switzerland; Sébastien Speierer, sebastien.speierer@epfl.ch, École Polytechnique Fédérale de Lausanne (EPFL), Lausanne, Switzerland; Wenzel Jakob, wenzel.jakob@epfl.ch, École Polytechnique Fédérale de Lausanne (EPFL), Lausanne, Switzerland.

During the adjoint phase, we must undo the effects of the path prefix to obtain the directional derivative at the *current* path vertex. This is in part done by subtracting matrices, which is unproblematic. However, the second part is more involved: we must multiply the directional radiance derivative by the inverse matrix $\mathbf{J}_{\text{ray}}^{-1}$. The matrix $\mathbf{J}_{\text{ray}} = \mathbf{A}_1 \cdot \mathbf{A}_2 \cdots \mathbf{A}_k$ is the product of all the local ray derivatives $\mathbf{A}_i$ encountered in the path prefix. The challenge is now that some of these matrices may not be invertible (e.g., when the associated vertex has a diffuse BRDF). The subsequent derivations will prove two claims:

(1) Adding noise to factors of the Jacobian product does not affect its expected value.
(2) This noise can be added in a way so that even an estimator based on matrix inverses remains unbiased.

Regarding claim (1), we must show that

$$\mathbb{E}[\mathbf{J}_{\text{ray}}] = \mathbb{E}[\mathbf{A}_1 \cdot \mathbf{A}_2 \cdots \mathbf{A}_k] = \mathbb{E}[(\mathbf{A}_1 + \varepsilon_1\mathbf{I}) \cdot (\mathbf{A}_2 + \varepsilon_2\mathbf{I}) \cdots (\mathbf{A}_k + \varepsilon_k\mathbf{I})] \tag{3}$$

where $\mathbf{A}_1, \ldots, \mathbf{A}_k$ are (fixed) matrices, and $\varepsilon_1, \ldots, \varepsilon_k \in \mathbb{R}$ are i.i.d. random variables with an expected value of 0. The equality follows directly from the linearity of the expected value and the statistical independence of the ε_k. Without loss of generality, we can consider the case of $k = 2$:

$$\mathbb{E}[(\mathbf{A}_1 + \varepsilon_1\mathbf{I})(\mathbf{A}_2 + \varepsilon_2\mathbf{I})] = \mathbb{E}[\mathbf{A}_1\mathbf{A}_2] + \underbrace{\mathbb{E}[\varepsilon_1\mathbf{I}\mathbf{A}_2]}_{=0} + \underbrace{\mathbb{E}[\mathbf{A}_1\varepsilon_2\mathbf{I}]}_{=0} + \underbrace{\mathbb{E}[\varepsilon_1\varepsilon_2\mathbf{I}]}_{=0} = \mathbb{E}[\mathbf{A}_1\mathbf{A}_2] \quad \square \tag{4}$$

The first equality uses the linearity of the expected value. We then see that all terms except for the first one become zero due since $\mathbb{E}[\varepsilon_k] = 0$.

Now for claim 2, we need to show that we can invert the prefix of that product without introducing bias. This is possible by using the same noise both during the forward accumulation and the backward pass. We can then invert the *same* matrices as during the forward pass. Under the assumption that the noisy matrices are then indeed invertible, the computation of the relevant Jacobian at bounce i can be written as:

$$\mathbb{E}\Big[\underbrace{[(\mathbf{A}_1 + \varepsilon_1\mathbf{I}) \cdots (\mathbf{A}_i + \varepsilon_i\mathbf{I})]^{-1}}_{\text{Path prefix}}(\mathbf{A}_1 + \varepsilon_1\mathbf{I}) \cdots (\mathbf{A}_i + \varepsilon_i\mathbf{I}) \cdot (\mathbf{A}_{i+1} + \varepsilon_{i+1}\mathbf{I}) \cdots (\mathbf{A}_k + \varepsilon_k\mathbf{I})\Big] \tag{5}$$

$$= \mathbb{E}\left[\cancel{\left[(\mathbf{A}_1 + \varepsilon_1\mathbf{I}) \cdots (\mathbf{A}_i + \varepsilon_i\mathbf{I})\right]^{-1}}\ \cancel{(\mathbf{A}_1 + \varepsilon_1\mathbf{I}) \cdots (\mathbf{A}_i + \varepsilon_i\mathbf{I})} \cdot (\mathbf{A}_{i+1} + \varepsilon_{i+1}\mathbf{I}) \cdots (\mathbf{A}_k + \varepsilon_k\mathbf{I})\right] \tag{6}$$

$$= \mathbb{E}\left[(\mathbf{A}_{i+1} + \varepsilon_{i+1}\mathbf{I}) \cdots (\mathbf{A}_k + \varepsilon_k\mathbf{I})\right] \quad \square \tag{7}$$

We see that we do *not* take the expected value of a matrix inverse, but the accumulated gradient sample values only ever include regular matrix products and additive noise. Therefore, the regularization in itself does not introduce bias. However, our system of adding uniform noise to every matrix is simplistic, and better strategies may be needed to guarantee good numerical conditioning.

3 DIFFERENTIABLE DELTA TRACKING

In this section, we provide the pseudocode for our differentiable delta tracking implementation. Listing 1 details the forward rendering code for a volumetric path tracer with next event estimation using ratio tracking [Novák et al. 2014]. For simplicity, the code does not support surfaces or multiple importance sampling. In Listing 2 we provide the corresponding adjoint version. In particular, the pseudocode details how we use PRB in a nested way to backpropagate through the transmittance estimation.

```
def sample_path_delta_tracking(ray):
    L = 0.0, β = 1.0
    x = ray.o
    while active path:
        # Sample the free-flight distance
        t = -log(1 - sampler.next_1d()) / σ̄
        x += t * ray.d

        # Continue in next iteration if a null interaction was sampled
        if sampler.next_1d() < 1 - σ_t(x) / σ̄:
            continue

        β *= albedo(x)

        # Sample emitter direction towards an emitter and emitter contribution
        # The L_e term is assumed to have not yet been attenuated by transmittance,
        # but includes all other factors (i.e., emitter PDF, phase function value, etc.)
        L_e, ω_e = sample_emitter(...)

        # Evaluate the transmittance to the emitter
        x_e = x, tr = 1.0
        while not reached emitter:
            t = -log(1 - sampler.next_1d()) / σ̄
            x_e += t * ω_e
            tr *= 1 - σ_t(x_e) / σ̄ # ratio tracking transmittance estimate

        L += tr * L_e
        ω' = sample_phase_function(...)
        ray = spawn_ray(x, ω')
    return L
```

Listing 1. Volumetric path tracer with delta tracking pseudocode.

```
def sample_path_delta_tracking_adjoint(ray, L, δL):
    β = 1.0
    x = ray.o
    while active path:
        t = -log(1 - sampler.next_1d()) / σ̄
        x += t * ray.d

        # Backpropagate for null interaction
        if sampler.next_1d() < 1 - σ_t(x) / σ̄:
            P_n = 1 - σ_t(x) / σ̄
            δ_π += backward_grad(P_n, δL * L / P_n)
            continue
        β *= albedo(x)

        # Backpropagate for real interaction
        P_s = σ_t(x) / σ̄
        δ_π += backward_grad(P_s * albedo(x), δL * L / (P_s * albedo(x)))

        L_e, ω_e = sample_emitter(...)
        x_e = x, tr = 1.0
        nee_sampler = sampler.copy() # create a copy of the random number generator with its current state
        while not reached emitter:
            t = -log(1 - nee_sampler.next_1d()) / σ̄
            x_e += t * ω_e
            tr *= 1 - σ_t(x_e) / σ̄

        x_e = x
        nee_sampler = sampler.copy()
        while not reached emitter:
            t = -log(1 - nee_sampler.next_1d()) / σ̄
            x_e += t * ω_e
            r = 1 - σ_t(x_e) / σ̄
            δ_π += backward_grad(r, δL * β * L_e / r)

        L -= tr * L_e
        ω' = sample_phase_function(...)
        ray = spawn_ray(x, ω')
    return δ_π
```

Listing 2. Pseudocode for the adjoint pass of the volumetric path tracer using delta tracking.

REFERENCES

Jan Novák, Andrew Selle, and Wojciech Jarosz. 2014. Residual Ratio Tracking for Estimating Attenuation in Participating Media. *ACM Transactions on Graphics (Proceedings of SIGGRAPH Asia)* 33, 6 (Nov. 2014).