Point cloud method for rendering BSSRDFs

Jeppe Revall Frisvad
Technical University of Denmark

Alessandro Dal Corso
Technical University of Denmark

March 2018

This is a short note on rendering a triangular mesh onto which we apply a BSSRDF model. The method works on arbitrary static triangular meshes and is unbiased.

We first define some quantities relative to our mesh. We consider a triangle mesh a set $M = \{T_i, i \in [0, N_\Delta - 1]\}$ composed of N_Δ triangles. Each triangle T_i is composed of three vertices:

$$T_i = \{\mathbf{v}_0^i, \mathbf{v}_1^i, \mathbf{v}_2^i\}$$

From these quantities it straightforward to define two derived quantities, the normal $\vec{n}_i$ and the area A_i of the triangle.

$$\vec{n}_i = \frac{(\mathbf{v}_1^i - \mathbf{v}_0^i) \times (\mathbf{v}_2^i - \mathbf{v}_0^i)}{\|(\mathbf{v}_1^i - \mathbf{v}_0^i) \times (\mathbf{v}_2^i - \mathbf{v}_0^i)\|}$$

$$A_i = \frac{1}{2}\|(\mathbf{v}_1^i - \mathbf{v}_0^i) \times (\mathbf{v}_2^i - \mathbf{v}_0^i)\|$$

Once we have defined our triangles, we can start describing out our solution.

Theoretically, we solve the standard rendering equation for BSSRDFs:

$$L(\mathbf{x}_o, \vec{\omega}_o) = \int_A \int_\Omega S(\mathbf{x}_i, \vec{\omega}_i, \mathbf{x}_o, \vec{\omega}_o) L_i(\mathbf{x}_i, \vec{\omega}_i)(\vec{n}_i \cdot \vec{\omega}_i) dA_i d\vec{\omega}_i$$

We now solve the integral with Monte Carlo integration with importance sampling. We take K samples from the area integral and one sample from the directional integral:

$$\begin{aligned} L(\mathbf{x}_o, \vec{\omega}_o) &\approx \sum_{k=1}^{K} \frac{S(\mathbf{x}_k, \vec{\omega}_k, \mathbf{x}_o, \vec{\omega}_o) L_i(\mathbf{x}_k, \vec{\omega}_k)(\vec{n}_k \cdot \vec{\omega}_k)}{\mathrm{pdf}(\vec{\omega}_k)\mathrm{pdf}(A_k)} = \\ &= \sum_{k=1}^{K} S(\mathbf{x}_k, \vec{\omega}_k, \mathbf{x}_o, \vec{\omega}_o) \Phi_{i,k}(\mathbf{x}_k, \vec{\omega}_k) \end{aligned} \tag{1}$$

The algorithm is composed of two phases. In the first phase, we generate the K samples on the surface of the model, storing for each position $\mathbf{x}_k$, direction $\vec{\omega}_k$ and incoming flux $\Phi_{i,k}$. In the second phase, we render the final image evaluating the contribution at each pixel.

In phase one, we produce K samples. The k-th sample is composed of a position $\mathbf{x}_k$, a direction $\vec{\omega}_k$ and a flux $\Phi_{i,k}$. First, we choose a triangle T_j. We choose the triangle uniformly, so given a random number $\xi \in [0,1)$, $j = \lfloor \xi N_\Delta \rfloor$. To get $\mathbf{x}_k$, we sample a point uniformly on the triangle T_j:

$$\mathbf{x}_k = (1-\sqrt{\xi_0})\mathbf{v}_0^j + (1-\xi_1)\sqrt{\xi_0}\mathbf{v}_1^j + \xi_1\sqrt{\xi_0}\mathbf{v}_2^j$$

Where $\xi_0, \xi_1 \in [0,1)$.

Now, the sampling direction $\vec{\omega}_k$ and the flux $\Phi_{i,k}$ need to be evaluated. We leave the choice of the sampling distribution for $\vec{\omega}_k$ to implementation, since it is not important for the method. From sampling the light source we obtain $\vec{\omega}_k$ and an irradiance E_k:

$$E_k(\mathbf{x}_k, \vec{\omega}_k) = \frac{L_i(\mathbf{x}_k, \vec{\omega}_k)(\vec{n}_k \cdot \vec{\omega}_k)}{\mathrm{pdf}(\vec{\omega}_k)}$$

We need the pdf of the area $\mathrm{pdf}(A_k)$. Since we sampled the triangle and then the point, we combine the two independent probabilities:

$$\mathrm{pdf}(A_k) = \mathrm{pdf}(T_k)\mathrm{pdf}(\mathbf{x}_k) = \frac{1}{N_\Delta A_j}$$

Putting it all together:

$$\Phi_{i,k}(\mathbf{x}_k, \vec{\omega}_k) = \frac{E_k(\mathbf{x}_k, \vec{\omega}_k)}{\mathrm{pdf}(A_k)} = N_\Delta A_j E_k(\mathbf{x}_k, \vec{\omega}_k)$$

Once we have stored this quantity in the sample, we are ready to proceed to the second and final phase. For each pixel, we ray trace a camera ray, obtaining a point $\mathbf{x}_o$ and a direction towards the camera $\vec{\omega}_o$. We consider the scattering medium with coefficients σ_a, σ_s, g and relative index of refraction η as a pure Fresnel interface with added scattering.

In our path tracing implementation, we perform russian roulette on the Fresnel coefficient R to choose reflected or refracted direction. In the case of reflection, we continue tracing the reflected ray. In the case of refraction, we calculate the final radiance as:

$$L_o(\mathbf{x}_o, \vec{\omega}_o) = L_e(\mathbf{x}_o, \vec{\omega}_o) + L_t(\mathbf{x}_o, \vec{\omega}_o) + L(\mathbf{x}_o, \vec{\omega}_o)$$

Where L_e is the emitted radiance from the medium, L_t is the radiance due to direct transmission and L comes from 1.

To efficiently render the model and save expensive BSSRDF evaluations, we do a last optimization. We evaluate the BSSRDF stochastically with probability $\exp(-\sigma_{tr}\|\mathbf{x}_o - \mathbf{x}_k\|)$, where $\sigma_{tr} = \sqrt{3\sigma_a(\sigma_a + (1-g)\sigma_s)}$ is the effective transport coefficient. So, only samples that are very close to the exit point $\mathbf{x}_o$ will be actually evaluated. In formulas:

$$L(\mathbf{x}_o, \vec{\omega}_o) \approx \sum_{k=1}^{K} S(\mathbf{x}_k, \vec{\omega}_k, \mathbf{x}_o, \vec{\omega}_o)\Phi_{i,k}(\mathbf{x}_k, \vec{\omega}_k)V(\xi, e^{-\sigma_{tr}\|\mathbf{x}_o - \mathbf{x}_k\|})\, e^{\sigma_{tr}\|\mathbf{x}_o - \mathbf{x}_k\|}$$

Where:

$$V(\xi, d) = \begin{cases} 1 & \text{if } \xi < d \\ 0 & \text{otherwise} \end{cases}$$

In case of a spectral RGB rendering, we pick σ_{tr} as the mean of the three rgb components.