IEICE TRANS. INF. & SYST., VOL.E98–D, NO.2 FEBRUARY 2015

PAPER

Efficient Hair Rendering under Dynamic, Low-Frequency Environmental Light Using Spherical Harmonics

Xiaoxiong XING[†], ***Member***, **Yoshinori DOBASHI**[†,††a)], ***Nonmember***, **Tsuyoshi YAMAMOTO**[†], **Yosuke KATSURA**[†††], ***Members***, ***and*** **Ken ANJYO**[††,†††], ***Nonmember***

SUMMARY We present an algorithm for efficient rendering of animated hair under a dynamic, low-frequency lighting environment. We use spherical harmonics (SH) to represent the environmental light. The transmittances between a point on a hair strand and the light sources are also represented by SH functions. Then, a convolution of SH functions and the scattering function of a hair strand is precomputed. This allows us to efficiently compute the intensity at a point on the hair. However, the computation of the transmittance is very time-consuming. We address this problem by using a voxel-based approach: the transmittance is computed by using a voxelized hair model. We further accelerate the computation by sampling the voxels. By using our method, we can render a hair model consisting of tens of thousands of hair strands at interactive frame rates.

key words: *hair rendering, spherical harmonics, environmental light, voxelization*

1. Introduction

Real-time hair rendering is a challenging topic in computer graphics due to the complexities of hair geometry and its scattering properties. Kajiya and Kay [1]'s model and Marschner et al. [2]'s model are two widely used models that capture the single scattering property of hair. Zinke et al. [3]'s model aims to simulate multiple scattering property of hair by approximating it with a global scattering function and a local scattering function.

The effect of environmental light needs to be taken into account in the natural environments. The intuitive way is to sample the environment to yield a number of directional lights, render the hair under these lights and then accumulate the results. However, a large number of sample lights are needed to produce realistic results, which makes it difficult to achieve real-time frame rates.

Recently, real-time rendering of dynamic hair has become possible by approximating the environmental light with a set of spherical radial basis functions (SRBFs) [4], [5]. Since the number of SRBF lights is far fewer than the number of directional lights generated from directly sampling, the SRBF-based methods are able to render the hair model more efficiently. However, these methods rely heavily on precomputation for the SRBF approximation of the environmental light. If the environmental light changes during run-time, these methods would fail, because the SRBF ligths cannot be changed accordingly.

In this paper we consider animated hair under dynamic, low-frequency environment lighting. We focus on the efficient computation of the single scattering effects using the Kajiya-Kay model. Self-Shadows of the hair are also taken into account to enhance the realism. The self-shadows are rendered by computing transmittances of light between points on hair strands and a light source. Since the environmental light is modeled as a set of directional light sources, the transmittance in our case accounts for the fraction of light arriving from all directions. The environmental light and the transmittance are thus represented as functions defined on a unit sphere. We start with approximating both the transmittance and the environmental light by using spherical harmonics. Similar to SRBF, spherical harmonics can be used to represent spherical functions. Though the use of spherical harmonics is limited to low-frequency functions, the SH representation can be computed in real-time. The intensities of the hair are then determined by simply computing dot products between the SH coefficient vectors for the environmental light and the transmittances. Although the environmental light is efficiently converted into an SH representation, computing an SH representation of the transmittance is time-consuming. To address this, we voxelize the hair volume and compute the transmittance at each voxel. We further accelerate the computation by sampling the voxels.

The paper is organized as follows. We review previous work that is relevant to our research in the next section and then provide an overview of our framework in Sect. 3. We present our algorithm through Sects. 4 and 5. Section 6 describes the implementation details. Finally, Sect. 7 concludes the paper.

2. Previous Work

This section reviews previous work that is relevant to our method.

Hair Scattering Functions The Kajiya-Kay model and Marschner's model are two widely used models. The Kajiya-Kay model divides the scattered light into the diffuse and specular components. The diffuse component is derived from a Lambert surface model. The specular component is

Manuscript received June 17, 2014.
Manuscript revised September 30, 2014.
Manuscript publicized October 29, 2014.

[†]The authors are with Hokkaido University, Sapporo-shi, 060–0808 Japan.
[††]The authors are with CREST, JST, Kawaguchi-shi, 332–0012 Japan.
[†††]The authors are with OLM Digital, Tokyo, 154–0023 Japan.
a) E-mail: doba@ime.ist.hokudai.ac.jp

DOI: 10.1587/transinf.2014EDP7203

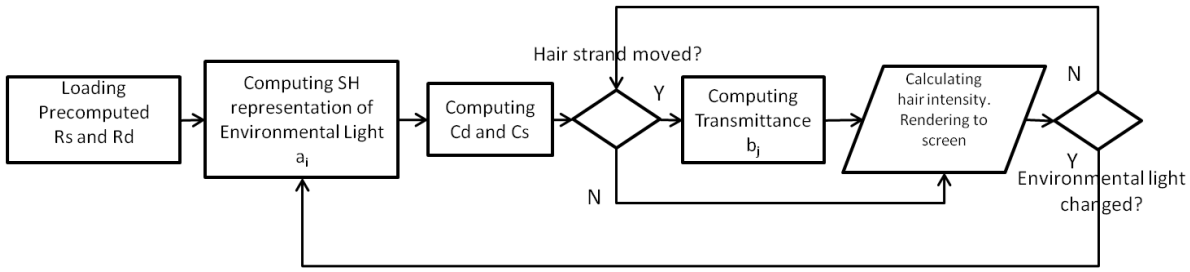


Fig. 1 Algorithm overview.

computed in a similar fashion to the Phong model, which is a general surface model by itself. Marschner's model is based on physical measurement and only accounts for specular reflection.

Hair Self-Shadowing A hair strand may absorb or scatter the light, and thus cast shadows on strands behind it. This kind of effect is called hair self-shadowing. Kim and Neumann [6] proposed the Opacity Shadow Maps (OSM) algorithm similar to the spirit of the shadow map. It divides the hair volume into several parts by using planes aligned perpendicular to the light direction, and compute the accumulated opacity at each plane. Sintorn and Assarsson [7] made it possible to render opacity maps in linear time. The Deep Opacity Map (DOM) algorithm [8], proposed by Yuksel and Keyser, removes the artifacts introduced by interpolation between parallel rectangle layers as in OSM. Different from the configuration of opacity maps used in OSM, DOM uses opacity map that follows the shape of hair. Sintorn and Assarsson [9] borrowed the technique of DOM and were able to reconstruct high resolution visibility functions. Yu et al. [10] proposed a framework for rendering complex scattering effects including volumetric shadows, transparency, and anti-aliasing.

Approximation of Environmental Light Tsai and Shih [11] proposed to use SRBFs to approximate the environmental light. The Gaussian SRBF kernel has been used in the approximation, and each one is described by two parameters, i.e. the center and the bandwidth. Given an environmental light, solving its SRBF approximation is formulated as an optimization problem to decide the parameters for each SRBF. The optimization process cannot be carried out with the parallel computation and is generally done offline. On the contrary, according to [12] the SH approximation of the environmental light can be accomplished in real-time. We take advantage of this feature in our algorithm to achieve rendering under dynamic environment. Mehta et al. [13]'s paper shows the use of spherical harmonics to approximate the diffuse term of Kajiya-Kay's model, which is a valuable reference for further refine our method.

Rendering Hair Under Environmental Light Ren et al. [4] first achieved real-time rendering of hair under environmental light. They approximate the environmental light by a set of SRBFs and thus convert the outgoing radiance integral into the sum of radiance contributions of all SRBF lights. For each SRBF light, they factor out the effective transmittance to represent the radiance integral as the product of the transmittance and the convolution of the SRBF light and the scattering function. In Ren et al.'s method, the convolution of the SRBF light and the scattering function has been pre-computed as a lookup table. Later, Xu et al. [5] derived a compact 1D circular Gaussian representation to model the scattering function and compute the rendering integral on the fly, thus eliminated the pre-computation of the convolution of the SRBF light and the scattering function. However, both methods use SRBFs to approximate the environmental light, so they cannot be applied to dynamic environment lighting.

3. Overview of Framework

We aim at rendering animated hair under dynamic environmental light. Figure 1 shows an overview of our method. We recompute the environmental light's SH representation only if it changes at run-time, and we recompute the transmittance for every vertex only if the hair animates. The transmittance is generally computed by using ray-tracing or the shadow map based algorithms, such as the DOM introduced in the previous section. Because of environmental light, the transmittance has to be computed for a set of sample directions. This process is time-consuming since a hair model typically consists of millions of vertices. However, the fact of the existing similarities among neighboring vertices inspires us to adopt a sampling strategy. We voxelize the hair model and use the voxel centers as sample locations to compute the transmittance. Transmittances of vertices are then computed as a weighted average of their nearby voxels. The SH representation of the environmental light and the transmittance are used in the final rendering pass to compute the intensity for a point on the hair model. We derive our rendering equation in Sect. 4 and describe the voxel-based approach for transmittance computation in Sect. 5.

4. Intensity Calculation Using Spherical Harmonics

We represent the hair strand with a set of connected line segments (see Fig. 2). The outgoing radiance along the viewing direction $\mathbf{v}$ at vertex $\mathbf{x}$ on a hair strand is expressed by:

$$I(\mathbf{x}, \mathbf{t}, \mathbf{v}) = \int T(\mathbf{x}, \omega) f(\mathbf{t}, \mathbf{v}, \omega) L(\omega) d\omega, \quad (1)$$

where $L(\omega)$ is the incoming radiance from direction ω, f is the scattering function, and $\mathbf{t}$ is the direction tangent to the

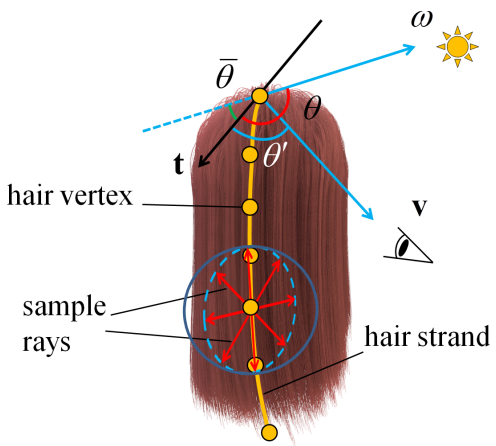


Fig. 2 The geometry of the hair and the Kajiya-Kay scattering function. **t** stands for the tangent direction, **v** stands for the viewing direction and ω stands for the lighting direction. Under environmental light, the transmittance is computed for a set of sample direction as indicated by the red arrows within the blue circle.

hair strand at $\mathbf{x}$. $T(\mathbf{x}, \omega)$ is the transmittance and is generally computed from the optical depth τ:

$$T(\mathbf{x}, \omega) = e^{-\kappa\tau(\mathbf{x},\omega)},$$
$$\tau(\mathbf{x}, \omega) = \int_0^l \rho(\omega, l')dl'.$$

$\rho(\omega, l')$ is the extinction coefficient that describes the percentage of light that is absorbed per unit distance at distance l' from the light source.

We approximate $L(\omega)$ and $T(\mathbf{x}, \omega)$ by using spherical harmonics $Y(\omega)$, i.e.,

$$L(\omega) \approx \sum_{i=0}^{m-1} a_i Y_i(\omega),$$
$$T(\mathbf{x}, \omega) \approx \sum_{j=0}^{n-1} b_j(\mathbf{x}) Y_j(\omega).$$

m and n are the number of spherical harmonics respectively used to represent the environmental light and the transmittance. a_i and $b_j(\mathbf{x})$ are the SH coefficients computed as:

$$a_i = \int L(\omega)Y_i(\omega)d\omega,$$
$$b_j(\mathbf{x}) = \int T(\mathbf{x}, \omega)Y_j(\omega)d\omega.$$

Note that the transmittance varies for every vertex, so we store the SH coefficients separately for each vertex. Expanding $L(\omega)$ and $T(\mathbf{x}, \omega)$ by their SH approximations, Eq. (1) then becomes:

$$I(\mathbf{x}, \mathbf{t}, \mathbf{v}) \approx \sum_{i=0}^{m-1}\sum_{j=0}^{n-1} a_i b_j(\mathbf{x}) R_{ij}(\mathbf{t}, \mathbf{v}), \quad (2)$$

where

$$R_{ij}(\mathbf{t}, \mathbf{v}) = \int f(\mathbf{t}, \mathbf{v}, \omega)Y_i(\omega)Y_j(\omega)d\omega.$$

R no longer depends on the hair geometry and the environmental light, so we can precompute it for a specified scattering function. We have applied the Kajiya-Kay shading model to the generation of all rendering results presented in this paper. The Kajiya-Kay shading model consists of a diffuse and a specular part, that is,

$$f(\mathbf{t}, \mathbf{v}, \omega) = K_d \sin\theta + K_s \cos^p(\bar{\theta} - \theta'). \quad (3)$$

K_d and K_s are the diffuse and specular reflectivity, respectively. p is a parameter controlling the sharpness of the highlight. θ is the angle between the light direction and the tangent direction (see Fig. 2). $\bar{\theta}$ is $\pi - \theta$. θ' is the angle between tangent direction and the viewing direction.

Following the expression of the Kajiya-Kay model, we can divide R into diffuse and specular components, R_d and R_s, respectively:

$$R_d(i, j, \mathbf{t}) = \int \sin\theta\, Y_i(\omega)Y_j(\omega)d\omega,$$
$$R_s(i, j, \mathbf{t}, \theta') = \int \cos^p(\bar{\theta} - \theta')Y_i(\omega)Y_j(\omega)d\omega.$$

We have omitted K_d and K_s here since they can be multiplied at the end of the rendering process. We stored the diffuse part in a table indexed by $(i, j, \mathbf{t})$ and the specular part in a table indexed by $(i, j, \mathbf{t}, \theta')$. Our implementation on the GPU evaluates each entry of the two tables simultaneously. In our experiments on a commodity graphics card, the computation time for R_d and R_s is less than 0.2 seconds.

Using the precomputed tables for R_d and R_s, we can efficiently compute the outgoing radiance I (Eq. (2)). However, its computational cost is $O(mn)$ and is still expensive. Since a_i is only affected by the environmental light, the product of a_i and R remains a constant unless the environmental light changes. Therefore, we first compute the products of a_i and R, and represent the products with $C_d(j, \mathbf{t})$ and $C_s(j, \mathbf{t}, \theta')$ respectively for the diffuse and specular parts, that is,

$$C_d(j, \mathbf{t}) = \sum_{i=0}^{m_d-1} a_i R_d(i, j, \mathbf{t}), \quad (4)$$
$$C_s(j, \mathbf{t}, \theta') = \sum_{i=0}^{m_s-1} a_i R_s(i, j, \mathbf{t}, \theta'). \quad (5)$$

m_d and m_s are the number of SH coefficients used for representing the environmental light. King [12] points out that 9 coefficients are sufficient for diffuse lighting, and $(\sqrt{p} + 1)^2$ coefficients are sufficient for specular lighting (where p appears in Eq. (3)). Therefore we approximate the environmental light with 9 SH coefficients when computing Eq. (4) and 64 when computing Eq. (5) (assuming $p = 49$ for convenience of computation and illustration purpose), so $m_d = 9$ and $m_s = 64$ in the above computation. For the transmittance, since it is a slowly changed function defined on the

unit sphere, we use 9 SH basis functions to approximate it. Then we store C_d and C_s into 2D texture arrays and send them into the hair rendering shader program where the intensity is computed as:

$$I(\mathbf{x},\mathbf{t},\mathbf{v}) = \sum_{j=0}^{n-1} b_j(\mathbf{x})(\, C_d(j,\mathbf{t}) + C_s(j,\mathbf{t},\theta')\,)$$

In this way, we avoid the necessity to compute the $O(mn)$ rendering equation directly in the shader program, and hence reduce the computational cost to $O(n)$.

In the next section, we describe an efficient voxel-based approach to obtain an SH representation of the transmittance.

5. Voxel-Based Computation of Transmittance

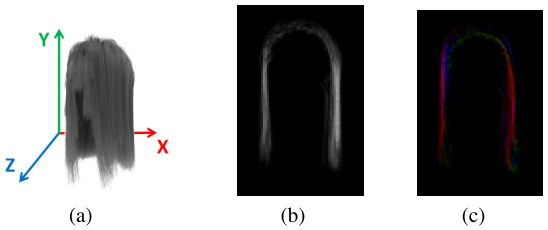
In order to compute the hair self-shadow due to an environmental light source, we need to compute the transmittances for a set of sample directions at each vertex. Computing the transmittance as seen from a sample direction requires us to compute intersections between the sample ray and the hair strands as shown in Fig. 2. However, since the number of strands of a model is very large, this process requires a extremely high computational cost.

Because the transmittance changes smoothly within the hair volume, our method voxelizes the hair volume and computes the transmittances for the voxels that contain hair strands (which we call non-empty voxels). The transmittance for each vertex is obtained by interpolating the transmittances computed at nearby voxels. To further accelerate the computation, our method computes the transmittances for a subset of the non-empty voxels and then interpolates them for the other voxels. Inspired by [14], we introduce the spatial gradient of the transmittance and use it at interpolation. The details are explained in the following.

Our method first converts the geometry of a hair into a volumetric representation following the approach in [15]. At each voxel, the density $\rho(\mathbf{x})$ of the hair strands, i.e., the number of hair strands passing through the voxel, is stored. Another volume that stores the gradient of the density, $\nabla\rho(\mathbf{x})$, is computed as:

$$\mathbf{u}\cdot\nabla\rho(\mathbf{x}) = (\,\rho(\mathbf{x}+\mathbf{u}) - \rho(\mathbf{x}-\mathbf{u})\,)/2. \tag{6}$$

$\mathbf{u}$ is a vector aligned with the positive direction of either the X, Y, or Z axis. Figure 3 shows slices (in the Z direction) from the density and the gradient volumes. Next, we select a subset of the non-empty voxels for computing the transmittance. The voxels are selected uniformly from the non-empty voxels, as shown in Fig. 4. The gray and green voxels are non-empty voxels and the green voxels are in the selected subset. The interval between the sample voxels are specified by the user. The centers of these sample voxels are then used as sample points $\mathbf{y}_i (i = 0, \cdots, N-1)$, and a set of sample rays are cast from each sample point. For each ray, the optical depth is computed as:



(a) (b) (c)

Fig. 3 (a) is the hair model. (b) shows a slice in the density volume where the density is represented by the intensity of pixel. (c) shows a slice in the gradient volume. The gradients in the X, Y, Z directions respectively are represented by the intensities of the red, green and blue channels.

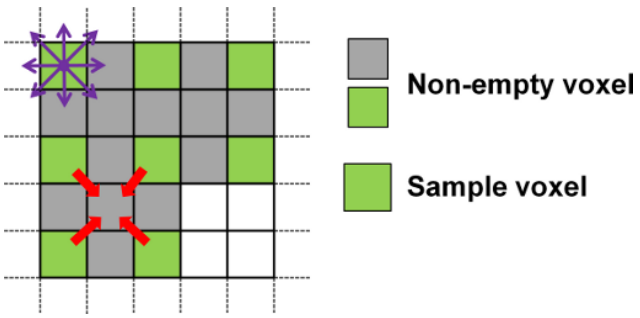


Fig. 4 This figure shows a slice in the grid after voxelization. The non-empty voxels are shown in the grey and green colors. We uniformly select the sample voxels (shown in green) and compute the transmittance by using ray casting as indicated by the purple arrows. After we have processed the sample voxels, we interpolate for the other non-empty voxels as indicated by the red arrows.

$$\tau(\mathbf{y}_i,\omega) = \int \rho(\mathbf{y}_i + t\omega)dt$$

The gradient of the optical depth is also computed by tracing the ray through the gradient volume, that is,

$$\nabla\tau(\mathbf{y}_i,\omega) = \nabla(\int \rho(\mathbf{y}_i + t\omega)dt) = \int \nabla\rho(\mathbf{y}_i + t\omega)dt.$$

In practice, $\tau(\mathbf{y}_i,\omega)$ and $\nabla\tau(\mathbf{y}_i,\omega)$ are computed by summing the values stored in the voxels that the sample ray ω intersects, in the density and the gradient volume respectively, namely,

$$\tau(\mathbf{y}_i,\omega) \approx \sum_t s\rho(\mathbf{v}_t),$$
$$\nabla\tau(\mathbf{y}_i,\omega) \approx \sum_t s\nabla\rho(\mathbf{v}_t).$$

s is the voxel length. $\mathbf{v}_t$ are the voxels that ω intersects and were found by using a 3D version of Bresenham's line drawing algorithm [16]. Alternatively, $\mathbf{v}_t$ can be found by using 3DDDA which is better than the implementation here.

The transmittance and its gradient are then calculated by using:

$$T(\mathbf{y}_i,\omega) = \exp(-\kappa\tau(\mathbf{y}_i,\omega)),$$
$$\nabla T(\mathbf{y}_i,\omega) = -\kappa\exp(-\kappa\tau(\mathbf{y}_i,\omega))\nabla\tau(\mathbf{y}_i,\omega),$$

In our experiment, we set the largest voxel density as the

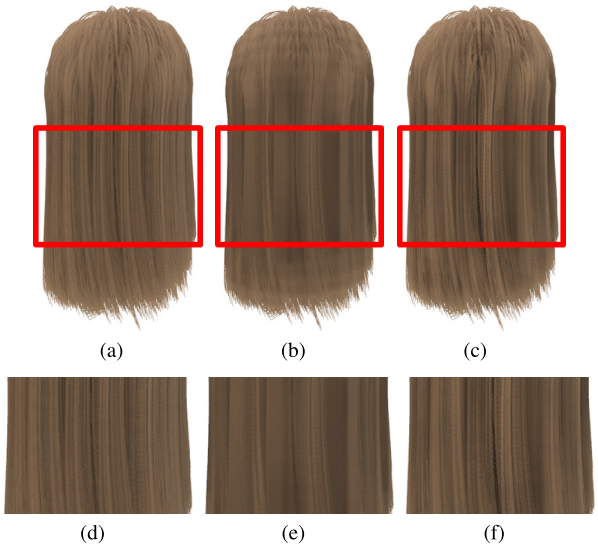


Fig. 5 (a) is the reference result generated by using all non-empty voxles in the transmittance computation. (b) and (c) are respectively generated by plain weighted average and with gradient from approximately 1/64 of the non-empty voxels. The interpolation range $\aleph(\mathbf{x})$ are the same in both cases. (d), (e) and (f) are the highlighted area in the upper images. Obviously, (e) suffers more from blurring than (f) does.

threshold for τ, and chose the value for κ such that when τ exceeds the threshold, T is approximately 0.

The transmittances and their gradients at each sample point are then converted into SH representations, resulting in SH coefficients $b_j(\mathbf{y}_i)$ and $\nabla b_j(\mathbf{y}_i)$, where

$$b_j(\mathbf{y}_i) = \int T(\mathbf{y}_i, \omega) Y_j(\omega) d\omega,$$
$$\nabla b_j(\mathbf{y}_i) = \int \nabla T(\mathbf{y}_i, \omega) Y_j(\omega) d\omega.$$

In the above procedure, we compute ∇b_j by first computing $\nabla\tau$ and then ∇T. We do in this way rather than directly compute them using the difference of b_j from neighbors because b_j is not a linear function of τ and b_j is only computed at sparsely sampled voxel locations.

The SH coefficients of the transmittance at each non-empty voxel $\mathbf{x}$ are finally obtained by using the weighted average with gradient from neighboring voxels:

$$b_j(\mathbf{x}) = \frac{\sum_{\mathbf{y}_i \in \aleph(\mathbf{x})} w_i (b_j(\mathbf{y}_i) + (\mathbf{x} - \mathbf{y}_i) \cdot \nabla b_j(\mathbf{y}_i))}{\sum_{\mathbf{y}_i \in \aleph(\mathbf{x})} w_i}, \quad (7)$$

where $\aleph(\mathbf{x})$ is the neighboring size, the user needs to assure it contains sample voxels for any $\mathbf{x}$. Here w_i is a weight that is inversely proportional to the distance between $\mathbf{x}$ and the nearby sample point. It is computed as $w_i = \exp(-\sigma\|\mathbf{x}-\mathbf{y}_i\|)$, and σ is chosen such that the weightings for voxels outside $\aleph(\mathbf{x})$ are approximately 0.

Figure 5 shows the effect of using gradient in the transmittance calculation. We compared the color value difference between (a) and (b),(c) in a vertex-by-vertex basis. The average color difference between (a) and (b) is 0.015, between (a) and (c) is 0.011 (color value in [0,1]), also from the image results we see (b) suffers more from blurring. Therefore, the weighted average with gradient performs better than the plain weighted average with regard to rendering quality, however, because the gradient volume is required to compute the gradient of transmittance, the plain weighted average is faster in frame rate. Table 1 shows the comparison of frame rates. The SH coefficients for every vertex are found by trilinear interpolation of coefficients from the 8-neighboring voxels.

Table 1 Comparison of frame rates (fps) between weighted average with and without gradient in the transmittance computation. Grid resolution: 99^3.

♯sample voxels/ ♯ non-empty voxels	1	1/9	1/27
With gradient	3.0	7.0	9.7
Without gradient	5.8	11.1	11.7

6. Results and Discussion

We used a computer with an Intel Core i7-2600k (CPU) and an NVIDIA GTX 580 (GPU). The hair model used in rendering consists of 3,359,072 vertices, 27,520 strands. Motions of hair were separately precomputed, and vertices positions were stored in advance.

6.1 Precomputation

$R_d(i, j, \mathbf{t})$ and $R_s(i, j, \mathbf{t}, \theta')$ are precomputed and stored in lookup tables. Each entry of R_d and R_s are approximately computed as:

$$R_d(i, j, \mathbf{t}) \approx \sum_k \sin\theta\, Y_i(\omega_k) Y_j(\omega_k) \Delta\omega_k,$$
$$R_s(i, j, \mathbf{t}, \theta') \approx \sum_k \cos^p(\bar{\theta} - \theta') Y_i(\omega_k) Y_j(\omega_k) \Delta\omega_k.$$

To determine an optimal sampling resolution for ω, we started with $3 \times 3 \times 6$ samples and gradually increased its amount during which we found that $7 \times 7 \times 6$ samples are sufficient for generating satisfying results. We determined the sampling resolution of $\mathbf{t}$ in the same fashion, which is also $7 \times 7 \times 6$. As for $\theta' \in [0, \pi]$, we sample it at a fixed step size. We have tested several step sizes and chose $\pi/8$ as the preferred size. The total storage for R_d and R_s is 18 MB.

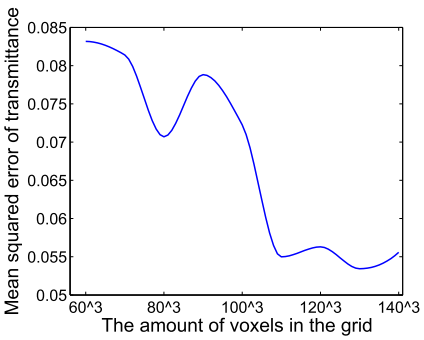
6.2 Grid Resolution

We used a regular grid in the voxelization. Each dimension of the grid contains 99 voxels. The number of sample rays in the computation of transmittance is $7 \times 7 \times 6$ which should be in accordance with the sampling resolution used in computing R_s and R_d.

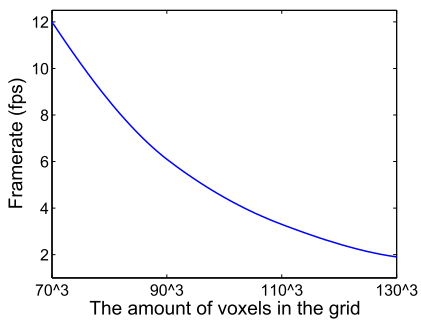
If we only need to render a static hair model, we can precompute the transmittances for all vertices and then use at final rendering. We applied the deep opacity maps algorithm [8] in the view of $7 \times 7 \times 6$ sample directions to the calculation of transmittances for each vertex. The transmittance value is first rendered into a framebuffer object and

(a) (b)

Fig. 6 (a) was rendered by using the transmittance computed in a preprocess. (b) was rendered by using transmittance that was computed by our voxel-based approach with 99^3 voxels.



(a)



(b)

Fig. 7 (a) shows the mean squared errors of transmittance values between vertices in the reference result and the results rendered with various grid resolutions using our method. The maximum transmittance is 1. We can see a descending trend of error along with the increasing of grid resolution. (b) shows the change of frame rate against grid resolution.

then fetched from there. This process is repeated for every vertex and for every viewing direction, and it takes several hours to complete. Figure 6 (a) shows the model rendered with precomputed transmittance. We use this result as a reference to compare with results rendered with various grid resolutions. The comparison is done by measuring the mean squared errors of transmittances between corresponding vertices in the reference and ours. From the diagrams in Fig. 7, we can see a descending trend of error along with the increasing of grid resolution, and the frame rate drops down with the increasing of grid resolution. The grid resolution is adjustable during runtime.

(a) (b) (c)

Fig. 8 (a) was rendered with a grid of 69^3 voxels at 7.8 fps. The artifacts caused by voxelization became evident due to low resolution. (b) was rendered with a grid of 99^3 voxels at 3.0 fps. (c) was rendered with a grid of 129^3 voxels but the frame rate drops to 1.3 fps.

We chose 99^3 as the grid resolution because of a trade-off between rendering quality and frame rate. Figure 8 shows the results rendered with various grid resolutions. Figure 9 shows the difference of two results rendered with different number of sample voxels.

6.3 Number of Spherical Harmonics

As an error metric of spherical harmonic approximation of the transmittance, we computed the mean squared errors between the original transmittance values and the transmittance values recovered from their spherical harmonic approximation with 9 coefficients, at non-empty voxel centers, which is 0.056. The mean squared error for the gradient of transmittance is 0.0032 (maximum transmittance is 1). The transmittance computation is independent of the environment map and only depends on the hair model.

At runtime, we use 64 SH basis functions to approximate the environmental light to compute the specular reflection (Eq. (5)). To demonstrate the effectiveness of using 64 SH basis functions, we generated a reference image computed by the following equation:

$$I(\mathbf{x},\mathbf{t},\mathbf{v}) \approx \sum_{j=0}^{n-1} b_j(\mathbf{x}) \int f(\mathbf{t},\mathbf{v},\omega) L(\omega) Y_j(\omega) d\omega, \qquad (8)$$

so that no SH approximation is applied to the environmental light. Figure 10 shows the comparison result. From this comparison, we can see that 64 SH basis functions are sufficient for generating the specular reflection.

6.4 Real-Time Performance

Figure 1 is an overview of our algorithm. R_d and R_s are loaded into memory after launching the program. The SH representation of the environmental light is then computed and updated at runtime upon change of environmental light. Further, C_d and C_s are computed accordingly. Both the two

(a) (b) (c)

Fig. 9 (a) was rendered at 3 fps by computing the transmittances for all non-empty voxels. (b) was rendered at 7 fps by computing the transmittances for 1/8 of the non-empty voxels and achieves similar quality as (a). (c) is the difference image between (a) and (b) after scaled by ten.

(a) (b)

Fig. 10 (a) is the reference image computed by following Eq. (8). (b) is computed by following Eq. (2). Only the specular reflection is included in this comparison.

Fig. 11 A screen shot from our video demo rendered at 2.9 fps.

computations are trivial compare with transmittance computation. When rendering dynamic hair, the transmittance of each sample voxel is updated each frame. Because our implementation parallelized this process, therefore, the real-time performance is proportional to the time it takes to compute the transmittances for one voxel, which further depends on the sampling resolution of transmittance, which equals that of ω when computing the integration of R_d and R_s. In our implementation, transmittance computation dominates the program's real-time performance. We did not use subsampling.

In the dynamic lighting situations, our current implementation achieves 2.9 fps for rendering the hair model when sampling all non-empty voxels. The overhead introduced by updating one frame of the environmental light is less than 50 ms and is nearly irrelevant to the size of the environment map (we use $256 \times 256 \times 6$ cube maps in the experiments). The previous methods require precomputation for the SRBF approximation of the environmental light, which takes 5–20 minutes for each frame as stated in [4]. The previous methods are not practical for dynamic environmental lighting. Figure 11 shows a screen shot from our video demo.

7. Conclusion and Future Work

In this paper, we proposed a framework for rendering animated hair under dynamic, low-frequency environmental lighting at interactive frame rates. Since the approximation of the environmental light with SH can be accomplished in real time, our method provides more flexibility than the existing methods [4] and [5] which are only suitable for rendering models under static environmental lighting.

In the future, we plan to extend our algorithm for the more physically-realistic Marschner's model. The challenge would be to precompute a larger table indexed by $(i, j, \mathbf{t}, \mathbf{v})$ which is required by Marschner's model rather than the current $(i, j, \mathbf{t}, \theta')$. The limitation of our work is that it is difficult to integrate the multiple scattering effect to our framework.

Acknowledgements

We would like to thank Dr. Paul Debevec for sharing the environment maps on his website.

References

[1] J.T. Kajiya and T.L. Kay, "Rendering fur with three dimensional textures," Proceedings of the 16th annual conference on Computer graphics and interactive techniques, SIGGRAPH 89, pp.271–280, New York, NY, USA, ACM, 1989.

[2] S.R. Marschner, H.W. Jensen, M. Cammarano, S. Worley, and P. Hanrahan, "Light scattering from human hair fibers," ACM SIGGRAPH 2003 Papers, SIGGRAPH 2003, pp.780–791, New York, NY, USA, ACM, 2003.

[3] A. Zinke, C. Yuksel, A. Weber, and J. Keyser, "Dual scattering approximation for fast multiple scattering in hair," ACM Trans. Graph., vol.27, no.3, pp.32:1–32:10, Aug. 2008.

[4] Z. Ren, K. Zhou, T. Li, W. Hua, and B. Guo, "Interactive hair rendering under environment lighting," ACM SIGGRAPH 2010 papers, SIGGRAPH 2010, pp.55:1–55:8, New York, NY, USA, ACM, 2010.

[5] K. Xu, L.Q. Ma, B. Ren, R. Wang, and S.M. Hu, "Interactive hair rendering and appearance editing under environment lighting," Proceedings of the 2011 SIGGRAPH Asia Conference, SIGGRAPH Asia 2011, pp.173:1–173:10, New York, NY, USA, ACM, 2011.
[6] T.Y. Kim and U. Neumann, "Opacity shadow maps," Proceedings of the 12th Eurographics Workshop on Rendering Techniques, pp.177–182, London, UK, UK, Springer-Verlag, 2001.
[7] E. Sintorn and U. Assarsson, "Real-time approximate sorting for self shadowing and transparency in hair rendering," Proceedings of the 2008 Symposium on Interactive 3D Graphics and Games, I3D '08, pp.157–162, New York, NY, USA, ACM, 2008.
[8] C. Yuksel and J. Keyser, "Deep opacity maps," Computer Graphics Forum (Proceedings of EUROGRAPHICS 2008), vol.27, no.2, pp.675–680, 2008.
[9] E. Sintorn and U. Assarsson, "Hair self shadowing and transparency depth ordering using occupancy maps," Proceedings of the 2009 Symposium on Interactive 3D Graphics and Games, I3D '09, pp.67–74, New York, NY, USA, ACM, 2009.
[10] X. Yu, J.C. Yang, J. Hensley, T. Harada, and J. Yu, "A framework for rendering complex scattering effects on hair," Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games, I3D '12, pp.111–118, New York, NY, USA, ACM, 2012.
[11] Y.T. Tsai and Z.C. Shih, "All-frequency precomputed radiance transfer using spherical radial basis functions and clustered tensor approximation," ACM Trans. Graph., vol.25, no.3, pp.967–976, July 2006.
[12] G. King, GPU Gems 2, ch. 10, Addison-Wesley Professional, 2005.
[13] S. Mehta, R. Ramamoorthi, M. Meyer, and C. Hery, "Analytic tangent irradiance environment maps for anisotropic surfaces," Computer Graphics Forum, vol.31, no.4, pp.1501–1508, June 2012.
[14] W. Jarosz, M. Zwicker, and H.W. Jensen, "Irradiance gradients in the presence of participating media and occlusions," Computer Graphics Forum (Proceedings of EGSR 2008), vol.27, no.4, pp.1087–1096, June 2008.
[15] J.T. Moon, B. Walter, and S. Marschner, "Efficient multiple scattering in hair using spherical harmonics," ACM SIGGRAPH 2008 papers, SIGGRAPH 2008, pp.31:1–31:7, New York, NY, USA, ACM, 2008.
[16] J.E. Bresenham, "Algorithm for computer control of a digital plotter," IBM Syst. J., vol.4, no.1, pp.25–30, March 1965.

Xiaoxiong Xing received his Bachelor degree from Beijing Normal University, China, in 2010. He is currently a PhD candidate at Hokkaido University.

Yoshinori Dobashi is an associate professor at Hokkaido University in the graduate school of engineering, Japan since 2000. His research interests center in computer graphics including lighting models. Dobashi received his BE, ME and Ph.D in Engineering in 1992, 1994, and 1997, respectively, from Hiroshima University. He worked at Hiroshima City University from 1997 to 2000 as a research associate.

Tsuyoshi Yamamoto is a Professor of Information Science and Technology at Hokkaido University. He received a degree of Doctor of Engineering in Electrical Engineering from Hokkaido University in 1986. He works in the fields of Computer Graphics, Symbolic Computation, Medical Image Processing and Information Media Technologies. He is a member of ACM SIGGRAPH, IEICE Japan, Information and Visual Media Society of Japan.

Yosuke Katsura is a Software Engineer in Computer Graphics field, and he is developing production tools and giving technical advise to artists. He has been working for OLM Digital, Inc. since 2006. His name has been credited in Pokemon the movies (2006 – 2012), Inazuma eleven the movie (2010, 2011), Lesson of the Evil (2012).

Ken Anjyo is Visual Effects/R&D Supervisor for OLM Digital. His research papers have been published in the major computer graphics journals and international conferences, such as IEEE CG&A and ACM SIGGRAPH. He has also been the project leader of "Mathematics for Expressive Image Synthesis" supported by the JST mathematics program since October 2010. He has served as organizer or committee member of leading international conferences, such as Digital Production Symposium 2012 co-chair and ACM SIGGRAPH ASIA 2009 Sketches and Posters chair.