ICIC Express Letters
Part B: Applications
Volume **7**, Number **4**, April **2016**

ICIC International ©2016 ISSN 2185-2766
pp. **805–811**

GPU-BASED RENDERING USING DISCRETE DIFFUSION MODEL AND CUBE ENVIRONMENT MAPPING

Gang Li[1,*] and Yifan Guo[2]

[1]College of Mathematics and Information Science
Zhengzhou University of Light Industry
No. 5, Dongfeng Road, Zhengzhou 450002, P. R. China
*Corresponding author: ligangzzuli@163.com

[2]Department of Computer
Henan Technical College of Construction
Zhengzhou 450000, P. R. China

Received October 2015; accepted January 2016

Abstract. *Realistic real-time rendering of translucent materials poses a challenging problem as a result of both requirements of photo-realistic and real-time rendering. In this paper we present GPU-based (Graphics Processing Unit) rendering using discrete diffusion model and cube environment mapping. We derive the discrete evaluation of diffusion model to give the photo-realistic rendering and fast GPU evaluation easily. Considering the influence to materials appearance from environment light, we design cube environment mapping on GPU and give a real-time realistic rendering. The result shows that we could acquire realistic appearance of translucent materials such as soft shadow and appearance details, which is more naturally integrated into the surrounding light.*

Keywords: Real-time rendering, Discrete diffusion model, Cube environment mapping, GPU evaluation

1. Introduction.

Realistic rendering of translucent materials poses a challenging problem in computer graphics. Many materials for realistic image synthesis, such as plants and fruits, beverages and food, many liquids and marble, exhibit Strong Subsurface Scattering (SSS) effects. Also when these materials are in some real scenes, their appearances will be influenced by surrounding light. So we have to simulate these effects to give a realistic rendering.

Since the material scatters light beneath the surface, we could in principle have to do a volume rendering, but the fact that we only need to worry about light emergent on the surface means that we can use an approximate analytical expression to describe the subsurface scattering under the surface. Several classic methods to simulate the subsurface scattering have been presented such as Dipole model [1], raytracing [2], multi-dipoles model [3] and spectral shading model [4]. However, these methods have to use long computation time to render the subsurface scattering. With the introduction of new hardware and software in computer graphics, the computational power has risen to a new level, where effects previously captured by photo-realistic rendering methods only, can be simulated in real-time applications as well.

Recently, some real-time algorithms to simulate SSS have already been introduced. Gauss kernel function [5], texture-space rendering [6] and screen-space rendering [7] are milestone algorithms among the methods of realistic rendering on GPU. However, these methods focus on the rendering using simple point or area lights for illuminating. Instead of using a certain number of light sources for illuminating a scene, it can be much more realistic to use environment lighting. Under complex environment lighting represented as Spherical Radial Basis Functions (SRBFs), Xu et al. [8] present an interactive algorithm for hair rendering and appearance editing. More recently, Kei et al. [9] propose an interactive rendering method of cloth fabrics under environment lighting; their GPU implementation enables interactive rendering of static cloth fabrics with dynamic viewpoints and lighting. Dahlin et al. [10] present a GPU-based rendering system based on the NVIDIA OptiX framework, enabling real-time raytracing of scenes illuminated with video environment maps. However, the rendering speed related above still needs to be improved to meet increasing requirements of faster interactive applications. In these applications such as 3D real-time games, the main challenge of rendering is not only to approximate the complex SSS to give a realistic looking appearance but also it should be quickly implemented and easily integrated with existing GPU pipelines.

We are motivated by the requirement of speed and implementation, and seek to render translucent objects in the real scene represented by environment map. We deduce the discretization of diffusion model, and give a realistic rendering for GPU implementation. Real world scenes usually have much more complex lighting conditions and environment mapping is an effective technique to capture complex lighting in GPU texture. So using GPU fragment shader, we simply look up the color of the environment in the direction of the viewing ray. Finally we can acquire a faster realistic rendering and give an easier implementation.

The rest of this paper is organized as follows. Discretization of diffusion model for GPU implementation is given in Section 2. Then fast cube environment mapping on GPU is showed in Section 3. Finally, experimental results are discussed in Section 4, and conclusions are drawn in Section 5.

2. **Discretization of Diffusion Model for GPU Implementation.** The BSSRDF (Bidirectional Scattering Surface Reflectance Distributed Function) can determine the light transport between any two rays intersecting a surface. As described in [1], the BSSRDF relates the differential reflected radiance, dL_r (at x_o in direction $\overrightarrow{\omega_o}$) to the differential incident flux, Φ_i (at x_i from direction $\overrightarrow{\omega_i}$) and can be written as:

$$S\left(x_i, \overrightarrow{\omega_i}, x_o, \overrightarrow{\omega_o}\right) = \frac{dL_r\left(x_o, \overrightarrow{\omega_o}\right)}{d\Phi_i\left(x_i, \overrightarrow{\omega_i}\right)} \tag{1}$$

From Equation (1), when light, with a direction $\overrightarrow{\omega_i}$, hits point x_i on a surface, the BSSRDF determines how much of this light is reflected from (another) point x_o in direction $\overrightarrow{\omega_o}$.

To find the reflected radiance, L_r, it is necessary to integrate Equation (1) over both area A and incoming directions

$$L_r\left(x_o, \overrightarrow{\omega_o}\right) = \int_A \int_{2\pi} S\left(x_i, \overrightarrow{\omega_i}, x_o, \overrightarrow{\omega_o}\right) L_i\left(x_i, \overrightarrow{\omega_i}\right) \left(\overrightarrow{\omega_i} \cdot N\right) d\overrightarrow{\omega_i} dA(x_i) \tag{2}$$

In highly scattering media, the radiation may scatter many times, which is known as multiple scattering as shown in Figure 1. Each scattering event blurs the light distribution; as a result the light distribution becomes quite uniform and tends to become isotropic.

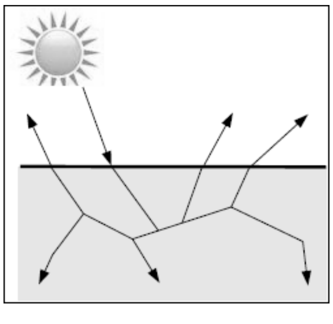


Figure 1. Multiple scattering

Multiple scattering plays a more crucial component in scattering events and mainly affects appearance of media.

Hence, we have to model the multiple scattering to give the complex effects of illumination on translucent objects. As we all know that the simulation of multiple scattering via traditional methods such as raytracing [2] is computation-intensive and time-consuming. Even the raytracing is improved by GPU rendering pipeline [10], the speed is still relatively low.

Considering the fast computation and real-time rendering, in this paper, we focus on simplifying the BSSRDF diffusion model and deduce the discretization of diffusion model for GPU implementation easily. Based on the reflected radiance according to Equation (2), the luminance calculation of multiple scattering could be acquired as

$$L_r\left(x_o, \overrightarrow{\omega_o}\right) = \int_A \int_{2\pi} S_{multiple}\left(x_i, \overrightarrow{\omega_i}, x_o, \overrightarrow{\omega_o}\right) L_i\left(x_i, \overrightarrow{\omega_i}\right)\left(\overrightarrow{\omega_i} \cdot N\right) d\overrightarrow{\omega_i} dA(x_i) \tag{3}$$

where $S_{multiple}$ is an item of multiple scattering, which can be evaluated further as

$$\begin{aligned} S_{multiple} &= \int_A \int_{2\pi} \frac{F_r\left(\overrightarrow{\omega_o}\right) R_d(x_i, x_o) F_r\left(\overrightarrow{\omega_i}\right)}{\pi} d\overrightarrow{\omega_i} dA(x_i) \\ &= \int_A \frac{F_r\left(\overrightarrow{\omega_o}\right) R_d(x_i, x_o)}{\pi} \int_{2\pi} F_r\left(\overrightarrow{\omega_i}\right) d\overrightarrow{\omega_i} dA(x_i) \end{aligned} \tag{4}$$

where F_r is the Fresnel reflectance term which computes the fraction of light reflected from an optically flat surface. R_d is the diffusion profile which could be simulated by sum-of-Gaussians in this paper.

Combining Equation (3) with Equation (4), we can acquire

$$L_r\left(x_o, \overrightarrow{\omega_o}\right) = \frac{F_r\left(\overrightarrow{\omega_o}\right)}{\pi} \int_A R_d(x_i, x_o) I\left(x_i, \overrightarrow{\omega_i}\right) dA(x_i) \tag{5}$$

where I is the irradiance map which can be represented as

$$I\left(x_i, \overrightarrow{\omega_i}\right) = \int_{2\pi} F_r\left(\overrightarrow{\omega_i}\right) L_i\left(x_i, \overrightarrow{\omega_i}\right)\left(\overrightarrow{\omega_i} \cdot N\right) d\overrightarrow{\omega_i} \tag{6}$$

As we all know that the rendering object is represented by the discrete 3D mesh and GPU rendering is for point based rendering, so we finally give the discretization of Equation (5) for GPU computation as follows:

$$L_r\left(x_o, \overrightarrow{\omega_o}\right) = \frac{F_r(\omega_o)}{\pi} \sum_{x_i \in VS} R_d(x_i, x_o) I(x_i) M(x_i) \tag{7}$$

where VS is the point set of 3D mesh model. M is the micro-facet distribution.

Using sum of four Gausses with variances υ_i and weights w_i, we can approximate the $R_d(r)$ as in

$$R_d(r) = \sum_{i=1}^{4} \omega_i G(\upsilon_i, r) \tag{8}$$

where r is the distance between x_i and x_o. The Gaussian of variance υ is defined as follows.

$$G(\upsilon, r) = \frac{1}{2\pi} e^{-\frac{r^2}{2\upsilon}} \tag{9}$$

Then we multiply the diffusion profile by an original irradiance map I to get a realistic appearance as follows.

$$I * R_d(r) = I * \left(\sum_{i=1}^{4} \omega_i G(\upsilon_i, r)\right) = \sum_{i=1}^{4} \omega_i I * G(\upsilon_i, r) \tag{10}$$

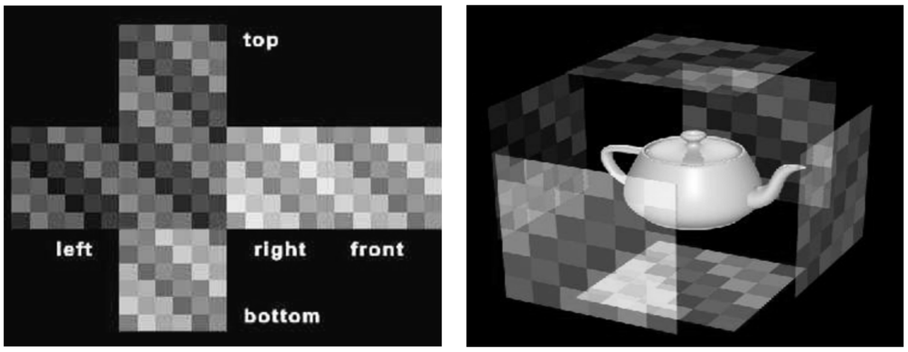
Because Gauss functions are simultaneously separable and radial symmetric, Gauss convolution at a wider stage can be computed from the result of a previous Gauss convolution, so the convolution of any two Gausses is another Gaussian as in

$$G(v_1, r) \times G(v_2, r) = G(v_1 + v_2, r) \tag{11}$$

Finally, using this special feature of Gauss convolution, we can implement a fast computation and give real-time rendering via GPU.

3. Fast Cube Environment Mapping on GPU.

Environment lighting is the lighting conditions surrounding the rendered object. In OpenGL when implementing environment lighting, there is a special kind of texture called cube map, which stores the images of distant environment. A cube map consists of six faces, and each of them has a 2D texture shown in Figure 2. We can sample a point on the cube with a 3D vector in order to get environment light intensity in a certain direction.



(a) Cube map (b) Teapot model in cube map

FIGURE 2. 3D model in cube environment map

Traditionally, considering one environment map with k texels, each texel can be thought of being a single light source. Therefore, for each surface point of the object the diffuse component can be computed as follows:

$$\textit{diffuseLight} = C \times \sum_{j=1\ldots k} \max\left(0, D_j \cdot N\right) L_j \tag{12}$$

where L_j is the light direction of texel j. N is the surface normal. D_j is the light direction. C is the surface albedo of diffuse surface.

It is obvious that if the number of texels k is large, computation of this sum in Equation (12) for each point of the surface of the object is really expensive. In this paper, we give a solution of this problem that we implement the precomputation of diffuse reflection via cube map texture of OpenGL. Our improvement is based on observations: firstly all surfaces with normal direction N will return the same value for the sum, and then the sum is dependent on just the lights in the scene and the surface normal. So we precompute the sum for any normal N and store result in a second environment map, indexed by surface normal. The second environment map is called *diffuse irradiance environment map*, which allows to illuminate translucent objects with arbitrarily complex lighting environments with quick lookup for cube map texture on GPU.

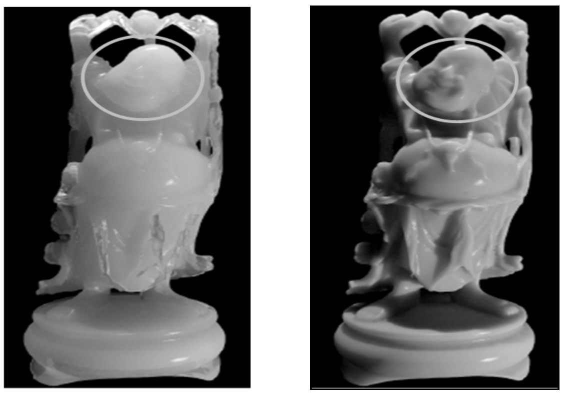
The key shader of $CubeMapping.vert$ and $CubeMapping.frag$ when implementing fast cube environment mapping on GPU is as follows.

```
/* Key GPU Shader of CubeMapping.vert */
gl_Position=pvmMatrix*vPos; /* Vertex Position to Clip Space */
tex_coord=vTexCoord.st;
vec3 e=normalize(vec3(ModelViewMatrix*vPos));
vec3 n=normalize(NormalMatrix*vNormal.xyz);
reflected=reflect(e, n); //reflection vector in eye and world coord
reflected=vec3(inverse(ViewMatrix)*vec4(reflected , 0.0));
```

```
/* Key GPU Shader of CubeMapping.frag */
//Perform a simple 2D texture look up.
vec3 base_color=texture2D(colorMap, tex_coord).rgb;
//Perform a cube map look up.
vec3 cube_color=textureCube(cubeMap, reflected);
//Write the final pixel.
gl_FragColor=vec4(mix(base_color , cube_color , reflect_factor), 1.0);
```

4. **Experiment Results.** We have implemented the fast realistic rendering using diffusion model and cube environment mapping on GPU. With Intel(R) Core(TM) i5-4770M CPU @ 3.40GHz and NVIDIA GeForce 9800GT, we have realized the method using GLSL shader and OpenGL programming in VS2014. We have achieved relatively high speed of rendering and our results clearly show the effective implementation of our technique.

Firstly, we give realistic rendering of marble budda using our method and BRDF (Bidirectional Reflectance Distribution Function). The number of budda triangles is about 78KB and the parameters weights w_i and variances v_i used are from Jensen et al.'s research [1]. Rendering effects of the marble budda are shown in Figure 3.



(a) Budda rendering via BRDF (b) Budda rendering via our method

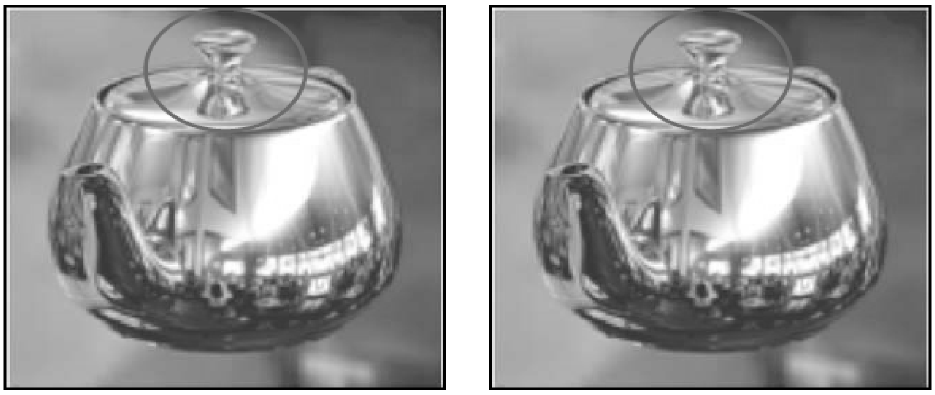
Figure 3. Comparison of budda rendering via two methods

From Figure 3, we could observe that not only the right result using our method gives a better translucency effects, but also the soft shadow and details such as face area can be rendered. So we give a much more realistic appearance over BRDF, which is truly accord with actual properties of marble budda.

Then, adding environment light we render the teapot containing about 46KB using our method and GPU raytracing [10], and give two rendering images as shown in Figure 4.

From Figure 4, using our method, we can render more clear reflection and depict a better shiny-looking effect, so that the model is more naturally integrated into the surrounding light.

To give an objective evaluation, we count the numbers of triangle and FPS (Frames Per Second) in the testing model for comparison with Monte Carlo raytracing [2] and GPU raytracing [10] shown in Table 1. From Table 1, we obtain a relatively less computation time as a result of precomputation and quick lookup. The main reason for this is that



(a) Rendering via GPU raytracing [10] (b) Rendering via our method

Figure 4. Comparison of teapot rendering via two methods

Table 1. FPS comparison via different methods

Models	*Triangles*	*Raytracing* [2]	*GPU raytracing* [10]	*Our method*
Budda	78*KB*	3*FPS*	51*FPS*	65*FPS*
Teapot	46*KB*	10*FPS*	75*FPS*	83*FPS*

environment mapping is essentially the process of pre-computing a texture map and then sampling texels from this texture during the rendering of a model. When increasing rendering points of translucent objects greatly, the difference of rendering time between our method and two comparison methods [2,10] is more obvious which shows that we are able to achieve faster speed while keeping realistic appearance.

Above all, we could get reasonable conclusion that our method can truly render the appearance of translucent objects in real scenes, and show their actual colors, soft shadow and appearance details.

5. Conclusions.

In this paper, an efficient method of realistic rendering has been proposed and implemented using discrete diffusion model and cube environment mapping on GPU. Experimental results show that we could show their actual colors and acquire realistic rendering of translucent materials. Some realistic effects such as soft shadow and appearance details can be simulated, which is more naturally integrated into the environment light. Also we obtain a relatively less computation time and higher FPS of rendering over Monte Carlo raytracing [2] and GPU raytracing [10].

Although the proposed approach has achieved a better rendering effect, it cannot express inter-object reflection very well. So how to approximate inter-object reflection to simulate global illumination is an interesting direction we are working on.

REFERENCES

[1] H. Jensen, S. Marschner, M. Levoy and P. Hanrahan, A practical model for subsurface light transport, *Proc. of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, pp.511-518, 2001.

[2] H. Jensen, Monte Carlo ray tracing, *ACM SIGGRAPH Course Notes*, vol.18, no.2, pp.163-196, 2003.

[3] C. Donner and H. Jensen, Light diffusion in multi-layered translucent materials, *ACM Trans. Graphics*, vol.24, no.3, pp.1032-1039, 2005.

[4] C. Donner and H. Jensen, A spectral BSSRDF for shading human skin, *Proc. of the Eurographics Conference on Rendering Techniques*, pp.409-417, 2006.

[5] G. Borshukov and J. Lewis, Realistic human face rendering for the matrix reloaded, *ACM SIGGRAPH Courses*, pp.13-23, 2005.

[6] E. d'Eon and D. Luebke, Advanced techniques for realistic real-time skin rendering, *GPU Gems 3*, vol.3, pp.293-347, 2007.

[7] J. Jimenez, V. Sundstedt and D. Gutierrez, Screen-space perceptual rendering of human skin, *ACM Trans. Applied Perception*, vol.6, no.4, p.3, 2009.

[8] K. Xu, L. Ma, B. Ren and A. H. Shimin, Interactive hair rendering and appearance editing under environment lighting, *ACM Trans. Graphics*, vol.30, no.6, pp.61-64, 2012.

[9] I. Kei, M. Kazutaka, D. Yoshinori and N. Tomoyuki, Interactive cloth rendering of microcylinder appearance model under environment lighting, *Computer Graphics Forum*, vol.33, no.2, 2014.

[10] J. Dahlin, D. Jonsson, M. Kok, T. B. Schon and J. Unger, Real-time video based lighting using GPU raytracing, *Proc. of the 22nd European Signal Processing Conference*, pp.1627-1631, 2014.