Image-Space Subsurface Scattering for Interactive Rendering of Deformable Translucent Objects

Article *in* IEEE engineering in medicine and biology magazine: the quarterly magazine of the Engineering in Medicine & Biology Society · March 2009

DOI: 10.1109/MCG.2009.11 · Source: PubMed

3 authors, including:

Musawir Shah
University of Central Florida
5 PUBLICATIONS **122** CITATIONS

SEE PROFILE

Sumanta Pattanaik
University of Central Florida
125 PUBLICATIONS **5,531** CITATIONS

SEE PROFILE

Image-Space Subsurface Scattering for Interactive Rendering of Translucent Objects

MUSAWIR A. SHAH
JAAKKO KONTTINEN
SUMANTA PATTANAIK
University of Central Florida

We present an image-space algorithm for real-time realistic rendering of translucent materials such as marble, wax, milk, etc. These materials exhibit strong subsurface scattering, which must be taken into account for synthesizing visually convincing images of translucent objects. Our method is able to capture subsurface scattering effects while maintaining interactive frame rates. It is based on dipole (or multipole) approximation of the BSSRDF model and it accounts for both the single and multiple scattering terms. Our algorithm operates in image-space, which decouples its runtime from the geometric complexity of the 3D model being rendered. It involves no pre-computation; a full evaluation of all the steps of the algorithm are performed every frame. Therefore, our method is able to support fully dynamic geometry, lighting, and viewing.

The main idea of our work is that it employs a "shooting approach" as opposed to the traditional "gathering approach" to evaluating the integral over the surface area for computing illumination due to multiple scattering. In doing so, implementing the integration computation on the GPU becomes very simple and natural. We use a dual light-view space technique, which enables us to effeciently compute the multiple scattering integral by considering only the significant point pairs on the surface of the object, and also eliminate artifacts due to projection from one space to another. Furthermore, our algorithm is also able to accommodate single scattering, which is generally left out in existing methods, in the same framework.

The results produced using our technique are visually comparable to those obtained using offline renderers but at rates orders of magnitude faster. Lastly, our algorithm runs entirely on the GPU and can be easily integrated into existing real-time rendering systems by inserting only three additional render passes.

Categories and Subject Descriptors: I.3 [**Computer Graphics**]: Three-Dimensional Graphics and Realism and Color, shading, shadowing, and texture

1. INTRODUCTION

Most real world materials exhibit translucency to some extent at an appropriate scale. Certain things that are highly scattering, such as marble and milk, give rise to unique optical phenomena that distinguish their visual appearance from other materials such as metals. Therefore it is an important goal in computer graphics to faithfuly represent these materials for realistic image synthesis. BRDFs (Bidirectional Reflectance Distribution Functions) are generally used to model the appearance of different materials, however they are not sufficient for translucent objects. This is because light undergoes subsurface scattering in translucent objects, and hence can enter and exit at different locations on the surface; whereas a BRDF assumes that light enters and exits from the same location. Due to this, some important visual effects associated with translucency, such as the appearance of backlit objects, cannot be captured with BRDFs. The scattering effects,

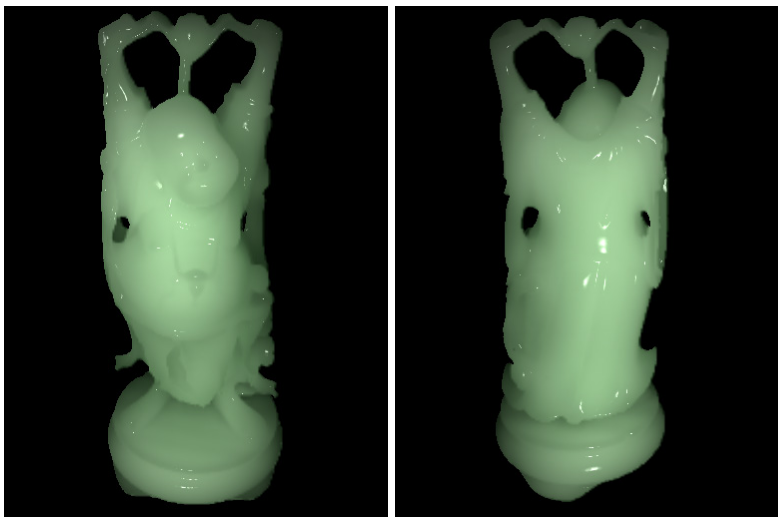

Fig. 1. The Happy Buddha model rendered using our algorithm with subsurface scattering properties of jade.

however, can be represented using the BSSRDF (Bidirectional Surface Scattering Reflectance Distribution Function) model, which takes into account both the angular and spatial distribution of light on the surface of an object. A BSSRDF is an 8-dimensional function that describes the light transport from one point to another for a given illumination and viewing direction.

A number of offline rendering methods that perform numerical simulation of subsurface scattering have been used for rendering of translucent materials. Although the results produced by these methods are extremely convincing, the computation involved is generally very expensive. With the recent introduction of the dipole BSSRDF model by Jensen et al. that employs the dipole source approximation for multiple scattering, a number of fast rendering algorithms have emerged. Our work, which is also inspired by the dipole BSSRDF model, is focused on developing a real-time algorithm for rendering translucent materials with no limitations on lighting, viewing, and object deformation while retaining the visual quality comparable to offline rendering results. Our main contribution is that we use the BSSRDF for evaluating subsurface scattering in an image-space framework, which gives us the additional advantage of supporting arbitrarily complex geometry. Furthermore, we show that our algorithm can be implemented entirely on the GPU thus freeing up the CPU for other computations.

The remainder of the paper is organized as follows: A brief survey of related work is presented in Section 2, followed by an overview of subsurface scattering theory and the BSSRDF model in Section 3. Our new algorithm is then explained in Section 4. The results are presented in Section 5 along with a discussion regarding the overall performance of our method and its limitations. We conclude with a summary of the ideas presented in the paper and provide directions for future research in Section 6.

2. PREVIOUS WORK

Subsurface scattering in translucent materials has been extensively studied in computer graphics. Impressive results have been produced from offline rendering techniques that simulate light transport in participating media. Although these approaches are able to faithfuly reproduce most of the subsurface scattering effects, they are generally slow. Faster algorithms have emerged after the introduction of the work by Jensen et al., which employs the dipole source approximation for multiple scattering in the BSSRDF model [Jensen et al. 2001]. A two pass heirarchical approach that significantly increased the speed of computation of the multiple scattering term was presented soon after [Jensen and Buhler 2002]. In this method, irradiance samples are taken on the surface of the object which are then organized into an octree data structure. This allows for heirarchical integration where the neighboring points are sampled densely and those further away are considered in groups. Recent methods have built up on this technique and accelerated the multiple scattering computation further using graphics hardware. These algorithms closely resemeble radiosity; the scattering event is treated as point-to-point [Lensch et al. 2002; Hao and Varshney 2004], point-to-patch [Mertens et al. 2003], and patch-to-patch [Carr et al. 2003] form-factor like transport. Although rendering at interactive frame-rates has been shown using these techniques, they are unable to efficiently handle environment lighting and the single scattering term is ignored completely. Furthermore in order to maintain interactivity, pre-computation is required for establishing the form-factor links in most cases. Unlike its counterparts, the technique presented by Mertens et al., which uses a hierarchical boundary element method to solve the subsurface scattering integral, is able to handle deformable geometry. They provide three different rendering modes of which the "on-the-fly" mode offers the most flexibility since it involves full evaluation from scratch. However it is also the slowest rendering mode producing an average framerate of about 9fps.

Subsurface scattering was included in the precomputed radiance transfer (PRT) framework by Sloan et al. in [Sloan et al. 2003]. They precompute and store the multiple scattering component of the BSSRDF as transport vectors in truncated lower order spherical harmonics (SH) basis, which are then combined with the SH coefficeints of the environment lighting function at render stage. Support for deformable objects was added to the PRT framework in [Sloan et al. 2005]. However, the major drawback is that due to the linear approximation in the SH basis, their method is limited to low frequency lighting signals only. This limitation was soon overcome by using wavelets. Support for all-frequency lighting was first introduced for the case of diffuse BRDF rendering by Ng et al. [Ng et al. 2003] and was recently adapted by Wang et al. [Wang et al. 2005] to include full BSSRDF rendering accounting for both the single and multiple scattering terms.

The goal of our work is to provide a GPU based real-time algorithm for rendering deformable translucent objects with dynamic lighting and viewing. Deviating from traditional convention of operating on geometry, we adopt an image-space approach, which allows us to efficiently employ the graphics hardware for the scattering computation as well as support objects of arbitrary geoemtric complexity without compromising real-time framerates. Image-space algorithms operate on

rasterized images of 3D scenes rather than the geometry. Due to this, they are very suitable for implementation on GPU. Recent advances in graphics hardware have enabled researchers to develop fast image-space algorithms for rendering complex optical effects such as reflection, refraction, and caustics [Szirmay-Kalos et al. 2005; Shah et al. 2006; Wyman 2005; Wyman and Davis 2006]. Francois et al. present a single scattering method for rendering layered material objects. The layers are stored as height fields in an image known as the *subsurface texture*, and single scattering is computed by ray marching through the layers and approximating the distance to the surface along the light direction at each step [Francois et al. 2006]. Mertens et al. present an image-space subsurface scattering algorithm that utilizes the dipole source approximation [Mertens et al. 2003]. They precompute a set of sample points for the area integration and evaluate the integral using multiple GPU passes. The key difference between this technique and our algorithm is that we adopt a "shooting" approach to evaluating the area integral in which the scattered light is shot from the light recieving points to the points visible from the camera, allowing for a more natural implementation on the GPU. As a result, our method does not involve a summation loop for shading each visible point since the accumulation is performed automatically in the GPU using blending. Furthermore, scattering in back lit objects in their method is hindered by the representation of the irradiance. They store the irradiance in a texture from the camera's view, in which case the back lit scattering effect cannot be achieved. Alternatively the irradiance texture can be stored from the light's view to obtain back lit scattering, however this results in artifacts due to stretching when the camera and light views are perpendicular. In our algorithm we use a dual light-camera view approach that enables us to capture back lit scattering and also eliminate stretching artifacts (see Section 4).

Similar to [Mertens et al. 2003], Dachsbacher and Stamminger present a subsurface scattering algorithm that employs translucent shadow maps [Dachsbacher and Stamminger 2003]. The integration is treated as a filtering operation at the visible points where the irradiance from nearby points is weighted and summed to compute the final exiting radiance. In order to speed up the rendering, they approximate the computation by separating the scattering into a local and a global response. At the global level, a heuristically determined filtering pattern is employed that uses the depth and spatial variation between two points to compute the dipole reflectance whereas at the local level the depth variation is ignored. Since no comparisons to reference rendering results are provided, it is difficult to verify the accuracy of the images presented. Furthermore, due to GPU storage limitations only directional lighting is considered.

In this paper, we present an image-space algorithm for real-time subsurface scattering in translucent materials. Our method also utilizes Jensen's dipole source approximation BSSRDF model and accounts for both the single and multiple scattering terms under point light illumination. Environment lighting is supported for multiple scattering only (see Section 4.2). The algorithm does not require any precomputation and is evaluated fully at every frame. We obtain results comparable to those produced using offline renderers at about 25 frames per second for arbitrarily complex objects.

3. BACKGROUND

Our algorithm employs the BSSRDF model for computing the subsurface scattering in a given homogenous object. BSSRDFs are one level higher up than BRDFs in the hierarchy of models for describing object appearance. A BRDF is a 4-dimensional function that describes the angular distribution of the outgoing radiance relative to the incoming radiance at a single point. However, translucency cannot be faithfully accounted for with such a representation. This is because translucent materials exhibit subsurface scattering, in which light entering at a certain point gets transmitted inside the medium and can exit at a different location. The BSSRDF is an 8-dimensional function that additionally describes the spatial distribution of light between two points on the surface. Therefore the exiting radiance at point x_o in direction ω_o is given by:

$$dL_o(x_o, \overrightarrow{\omega_o}) = S(x_i, \overrightarrow{\omega_i}, x_o, \overrightarrow{\omega_o}) d\Phi(x_i, \overrightarrow{\omega_i}) \tag{1}$$

$$L_o(x_o, \overrightarrow{\omega_o}) = \int_A \int_{2\pi} S(x_i, \overrightarrow{\omega_i}, x_o, \overrightarrow{\omega_o}) L_i(x_i, \overrightarrow{\omega_i}) (\overrightarrow{n_i} \cdot \overrightarrow{\omega_i}) d\omega_i dA(x_i) \tag{2}$$

The various symbols used throughout this paper are defined in Table I. The BSSRDF, $S(x_i, \overrightarrow{\omega_i}, x_o, \overrightarrow{\omega_o})$, is generally considered as a sum of a single scattering term $S^{(1)}$ and a diffuse multiple scattering term S_d:

$$S(x_i, \overrightarrow{\omega_i}, x_o, \overrightarrow{\omega_o}) = S^{(1)}(x_i, \overrightarrow{\omega_i}, x_o, \overrightarrow{\omega_o}) + S_d(x_i, \overrightarrow{\omega_i}, x_o, \overrightarrow{\omega_o}) \tag{3}$$

It has been shown that for highly scattering translucent materials it is sufficient to only consider the diffuse multiple scattering term of the BSSRDF. Most of the current interactive BSSRDF rendering techniques therefore do not account for the single scattering term which enables them to simplify computation and consequently increase speed.

3.1 Single Scattering

The outgoing radiance contribution from the single and multiple scattering terms can be considered separately. The single scattering process is depicted in Figure

Symbol	Description
$\overrightarrow{\omega_i}$	incident light direction
$\overrightarrow{\omega_o}$	exiting light direction
n_i	surface normal
g	average cosine of scattering angle
σ_s	scattering coefficient
σ_a	absorption coefficient
σ'_s	reduced scattering coefficient $= (1-g)\sigma_s$
σ_t	extinction coefficient $= \sigma_a + \sigma_s$
σ'_t	reduced extinction coefficient $= \sigma_a + \sigma'_s$
η	relative refraction index
$F_t(\eta, \overrightarrow{\omega})$	Fresnel transmittance
$p(\overrightarrow{\omega_i}, \overrightarrow{\omega_o})$	phase function

Table I. Symbols used in the multiple scattering computation using the dipole source BSSRDF model.

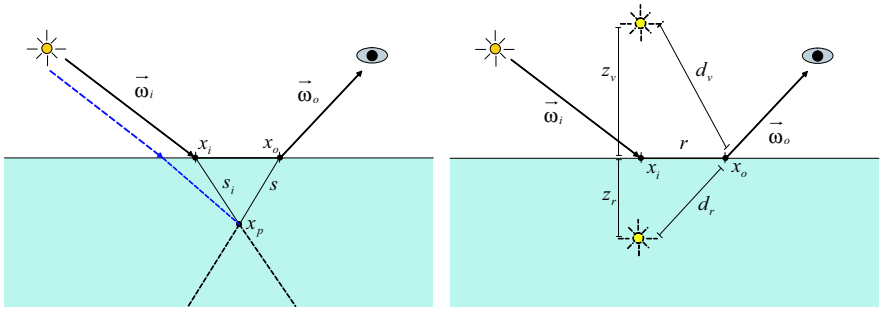

Fig. 2. (a) Radiance along $\overrightarrow{\omega_o'}$ due to single scattering at a point x_p . Note that the light direction $\overrightarrow{\omega_i}$ shown in blue does not undergo refraction and this is used instead of $\overrightarrow{\omega_i'}$ in the computation for practical purposes. (b) Dipole source approximation for multiple scattering.

2(a). The view ray incident at a point x_o in direction $\overrightarrow{\omega_o}$ is transmitted into the material in the refraction direction $\overrightarrow{\omega_o'}$. Single scattering is obtained by accumulating the amount of light transmitted to sample points along $\overrightarrow{\omega_o'}$. This is given by the following integral:

$$L_o^{(1)}(x_o, \overrightarrow{\omega_o}) = \sigma_s \int_{2\pi} \int_0^{\infty} F\ p(\overrightarrow{\omega_i'}, \overrightarrow{\omega_o'}) e^{-\sigma_t(s_i'+s)} L(x_i, \overrightarrow{\omega_i}) ds d\overrightarrow{\omega_i} \tag{4}$$

where σ_s is the scattering coefficient, $\sigma_t = \sigma_s + \sigma_a$ is the extinction coefficient and σ_a is the absorption coefficient of the translucent material. p is the phase function and F is the product of the Fresnel transmittance terms at x_i of light entering the material and at x_o of light leaving the material towards the viewer: $F = F_t(\eta, \overrightarrow{\omega_i}) \cdot F_t(\eta, \overrightarrow{\omega_o})$. s_i' and s are the distances that light travels inside the medium along directions $\overrightarrow{\omega_i'}$ and $\overrightarrow{\omega_o'}$ respectively. If the single scattering integral is carried out by taking samples x_p along $\overrightarrow{\omega_o'}$, it is not trivial to find the point of refraction x_i (in order to compute distance s_i') on the surface of an arbitrary object such that the refracted illumination ray passes through x_p. Therefore in practice the refraction of the illumination ray is ignored to simplify the computation by substituting $\overrightarrow{\omega_i}$ for $\overrightarrow{\omega_i'}$.

3.2 Multiple Scattering

The multiple scattering term accounts for the diffusion of light inside the material. Although an analytic solution for multiple scattering is not known, it can be estimated using the dipole source approximation defined by Jensen et al as follows:

$$S_d(x_i, \overrightarrow{\omega_i}, x_o, \overrightarrow{\omega_o}) = \frac{1}{\pi} F_t(\eta, \overrightarrow{\omega_i}) R_d(||x_i - x_o||) F_t(\eta, \overrightarrow{\omega_o}) \tag{5}$$

where R_d is computed by:

$$R_d(r) = \frac{\alpha'}{4\pi} \left[z_r \left(\sigma_{tr} + \frac{1}{d_r} \right) \frac{e^{-\sigma_{tr} d_r}}{d_r^2} + z_v \left(\sigma_{tr} + \frac{1}{d_v} \right) \frac{e^{-\sigma_{tr} d_v}}{d_v^2} \right] \tag{6}$$

where $\alpha' = \sigma'_s/\sigma'_t$ is the reduced albedo obtained from the reduced scattering and extinction coefficients: $\sigma'_s = (1-g)\sigma_s$ and $\sigma'_t = \sigma_a + \sigma'_s$, and g is the mean cosine of the scattering angle. $\sigma_{tr} = \sqrt{3\sigma_a\sigma'_t}$ is the effective extinction coefficient. The dipole approximation treats the multiple scattering contribution to a point x_o of an incoming light ray at the point x_i as illumination from a pair of real-virtual light sources placed below and above the surface at distances z_r and z_v respectively (Figure 2(b)). d_r and d_v are distances from x_o to the real and virtual dipole lights. These distances are computed as follows:

$$d_r = \sqrt{r^2 + z_r^2} \tag{7}$$

$$d_v = \sqrt{r^2 + z_v^2} \tag{8}$$

$$r = ||x_o - x_i|| \tag{9}$$

$$z_r = \frac{1}{\sigma'_t} \tag{10}$$

$$z_v = z_r(1 + \frac{4A}{3}) \tag{11}$$

$$A = \frac{(1 + F_{dr})}{(1 - F_{dr})} \tag{12}$$

$$F_{dr} = \frac{-1.440}{\eta^2} + \frac{0.710}{\eta} + 0.0668 + 0.0636\eta \tag{13}$$

Note that R_d is simply a function of the distance between the incoming and outgoing points, x_i and x_o. Therefore, for a given material with known absorption and scattering coefficients, R_d can be computed and stored as a lookup table indexed by distance. However, computing the outgoing radiance at x_o involves evaluating $R_d(||x_o - x_i||)$ for all sample points x_i over the entire surface of the object. Thus the cost of the final rendering is quadratic in the number of sample points, which can be quite expensive. Most recent work has focused on finding ways for performing this integration process efficiently by using heirarchical data structures, pre-computation, etc. It is also worth noting that the procedures for rendering single scattering and the dipole approximation multiple scattering are fairly different. Therefore, they generally require separate evaluation and incur added computational cost. This is also one of the reasons why some existing methods choose to ignore the single scattering term altogether, specially for highly scattering materials in which the single scattering term is quite low. Our algorithm is able to support the evaluation of both the single and multiple scattering terms in a single step. This is a direct consequence of our "shooting" approach to evaluating the integral as opposed to the traditional "gathering" approach. Details of our rendering algorithm are presented in the following section.

4. OUR ALGORITHM

Our algorithm is based on the fact that we interpret the area integration problem required for subsurface scattering as a "shooting" process rather than a "gathering" process. In order to solve Equation 2 to find the exiting radiance at x_o, one must take samples points x_i over the surface and "gather" the scattering contribution

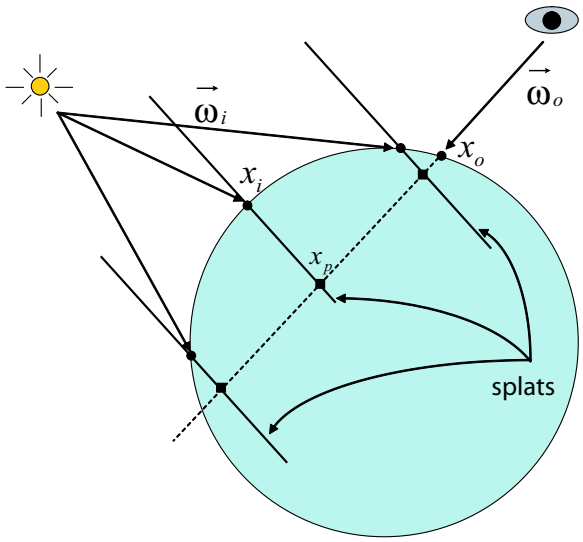

Fig. 3. Diagram of our rendering algorithm. The points x_p on the individual splats project on to the visible surface points x_o. The multiple scattering term is evaluated at each x_p by taking the distance between x_o and the corresponding x_i. The results are accumulated using additive alpha blending and stored at x_o.

to x_o from each of those points. In our approach, we take sample points on the surface visible from the light source and "shoot" the scattering contribution to all the points visible to the viewer within the effective scattering range from each point. The end result is the same; each point on the surface that is rendered will receive the scattering contribution from all the points that influence it. However our approach has two main advantages. First, we only compute scattering between significant point pairs, i.e. pairs of shooting and gathering points such that the the shooting points must have incident light to scatter, and the recieving points must be visible to the viewer. Secondly, treating the integration computation in this fashion allows for a very simple and natural implementation on the GPU using additive alpha blending.

We first describe our rendering algorithm for a single object under point light illumination. Environment lighting is also supported and is described later in Section 4.2. Furthermore, since our algorithm operates in image-space, in Section 4.3 we show that multiple objects with different scattering properties are rendered simultaneously without any complication or any additional cost.

The algorithm consists of the following steps:

Compute dipole approximation lookup table: Equation 6 is computed and stored in a texture for a number of prescribed parameter values. The range of the distance values used for computing R_d depends on the absorption and scattering coefficients of the material being rendered. The maximum distance is chosen such that the value of R_d is below some threshold. Note that this step is not performed every frame since the lookup table changes only with the material's scattering properties. Thus for a given material, this lookup table is fixed.

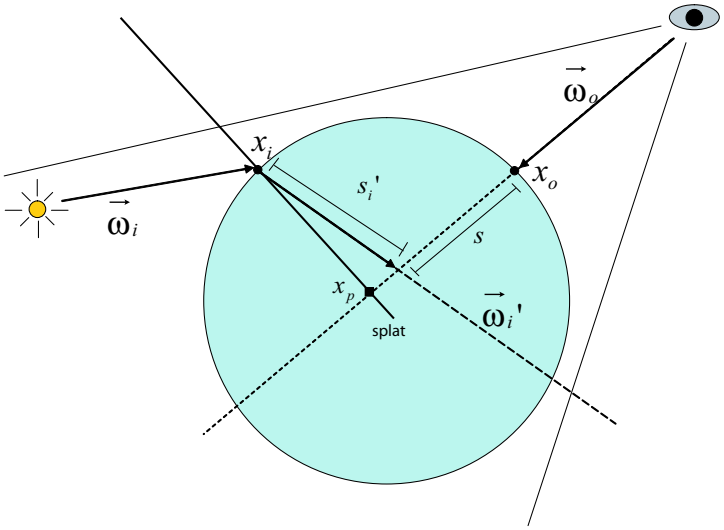

Fig. 4. Computation of the single scattering term in our algorithm.

Create scatter texture: The scatter texture contains the illumination due to subsurface scattering towards the visible surface of the object. The object is first rendered from the light's view in order to obtain sample points x_i on the surface that receive light. Light is then distributed, or "shot", from these points to the points visible to the viewer. This is done by rendering screen-aligned quads, also known as "splats", centered at each x_i (see Figure 3). The size of the splats corresponds to the maximum distance for which the dipole term R_d is computed in the previous step. Every pixel x_p on the quad is projected into the camera's view space to obtain the corresponding visible point x_o on the surface. The dipole approximation lookup table is then queried using the distance between x_o and x_i, and the value is modulated with the incident light and Fresnel terms to obtain the amount of light scattered from x_i to x_o due to multiple scattering. Additive alpha blending is used to accumulate the scattered light to x_o from all x_i. It is important to note that the scatter texture is rendered from and stored in the camera's view space as opposed to the light's view space. The reason for this dual-space approach is that the light's view captures all the points that receive light. However, light can be scattered to other points on the surface of the object that are not visible from the light's view. Therefore if the scatter texture is stored in the light's view space, those points will not be accounted for and important visual effects such as the scattering in back lit objects will not be attained. On the other hand, if the scatter texture is stored in the camera's view space, scattering to all the visible points will be computed. Although some points that could potentially receive light due to scattering are still left out, they are insignificant since they are not visible to the viewer. Furthermore, other than accurately accounting for the scattering, this dual-space representation also helps in eliminating stretching artifacts that occur when points are projected from camera's view space to the light's view like in shadow mapping. In our algorithm, even though the points x_i are sampled in

the light's view, the scattering is performed by rendering splats perpendicularly aligned with the camera's view. In doing so, stretching artifacts are prevented since no conversion from one space to the other is involved.

The single scattering term is also computed in this same step (see Figure 4). The vector $x_p - x_i$ is projected onto the refracted light direction $\overrightarrow{\omega_i'}$ and the distance s_i' is computed. The single scattering contribution from x_i to x_o is then computed as:

$$L_o^{(1)'}(x_p, x_o) = \sigma_s F \; p(\overrightarrow{\omega_i'}, \overrightarrow{\omega_o}) e^{-\sigma_t(s_i' + ||x_p - x_o||)} L(x_i, \overrightarrow{\omega_i'})(\overrightarrow{n_i} \cdot \overrightarrow{\omega_i'}) ds \qquad (14)$$

Note that this computation may not be completely accurate. It assumes that the view ray $\overrightarrow{\omega_o}$ and the refracted light ray $\overrightarrow{\omega_i'}$ intersect. Mathematically the intersection of 3D rays is defined only if they are coplanar. In practice, we compare the distance between two lines with a threshold for deciding if the two rays intersect.

Final rendering with subsurface scattering: In this step the scatter texture is simply mapped onto the object and specular highlights are added in.

4.1 Algorithm Implementation Steps

Since our algorithm runs entirely on the GPU, it is useful to describe it in terms of render passes. The algorithm consists of three render passes to compute the subsurface scattering texture, and a final render pass in which this texture is mapped onto the object.

Pass 1: Render object from light's view. In the pixel shader, output 3D world space positions and normals to a texture. This gives a set of "shooting" points x_i that receive light.

Pass 2: Render object from camera's view. In the pixel shader, output 3D world positions to a texture. This texture is used to lookup the points x_o on the surface visible to the viewer by projecting the points on the splat x_p in the camera's view space.

Pass 3: From the camera's view, render $n \times n$ screen-aligned quads (or splats) to create the scatter texture, where $n \times n$ is the resolution of the texture rendered in Pass 1. Each splat is centered at the corresponding x_i and is parallel to the camera's view plane. Note that the splats are positioned dynamically in the vertex shader. This requires a texture lookup in the vertex shader which is supported by shader model 3.0 and above. Alternatively, the render-to-vertex buffer extension of OpenGL can be used to output splat positions. In the pixel shader, the single and multiple scattering terms at each pixel x_p are computed as shown earlier. Additive alpha blending must be enabled for this pass in order to accumulate the scattering contribution from each splat.

Finally, the object is rendered with subsurface scattering using the scatter texture and the specular reflections are added in.

4.2 Environment Lighting

So far we have discussed our subsurface scattering algorithm in the context of point light illumination. However, it can be easily extended to support environment lighting. There are two key differences: (i) the incident light at each point on the surface of the object has a hemi-spherical distribution instead of just a single illumination direction, and (ii) all points on the surface can potentially receive

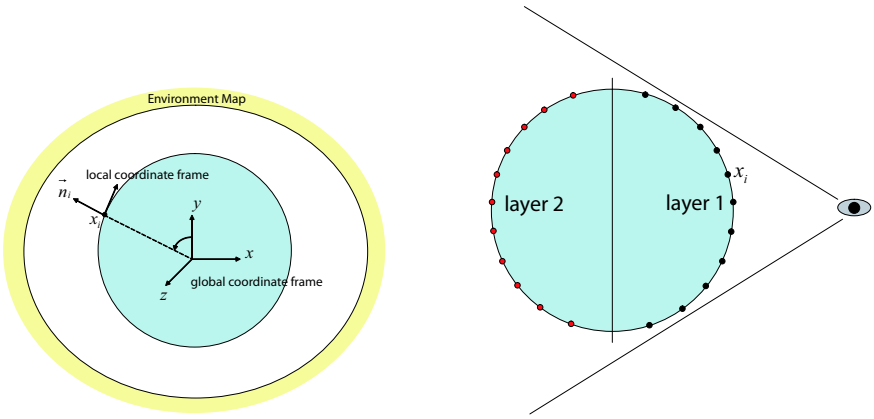

Fig. 5. (a) Spherical harmonic coefficients are stored in an environment map after prerotating them into the local coordinate frame formed by the direction vector of the corresponding pixel in the environment map. (b) Depth peeling to capture all potential points scattering points, x_i.

light as opposed to the point light illumination case in which only the points visible from the light's view can receive light. The first issue is managed using spherical harmonics to represent the environment lighting. Spherical harmonics are a set of orthnormal basis functions defined over the sphere. The basis functions are defined as:

$$Y_l^m(\theta,\phi) = K_l^m e^{im\phi} P_l^{|m|}(cos\theta), l \in N, -l \leq m \leq l \tag{15}$$

where P_l^m are the associated Legendre polynomials and K_l^m are the normalization constants

$$K_l^m = \sqrt{\frac{(2l+1)(l-|m|)!}{4\pi(l+|m|)!}} \tag{16}$$

Note that Equation 15 gives a complex basis. In order to obtain the real valued basis the following transformation is applied:

$$y_l^m = \begin{cases} \sqrt{2}Re(Y_l^m), & m > 0 \\ \sqrt{2}Im(Y_l^m), & m < 0 \\ Y_l^0, & m = 0 \end{cases} = \begin{cases} \sqrt{2}K_l^m cos(m\phi)P_l^m(cos\theta), & m > 0 \\ \sqrt{2}K_l^m sin(-m\phi)P_l^{-m}(cos\theta), & m < 0 \\ K_l^0 P_l^0(cos\theta), & m = 0 \end{cases} \tag{17}$$

To compute the transmitted light at a given point x_i from the environment the following integral must be evaluated:

$$L(x_i) = \int_{2\pi} L_i(\overrightarrow{\omega_i}) F_t(\eta, \overrightarrow{\omega_i})(\overrightarrow{n_i} \cdot \overrightarrow{\omega_i}) d\overrightarrow{\omega_i} \tag{18}$$

where $L_i(\overrightarrow{\omega_i})$ is the value of the environment lighting function in direction $\overrightarrow{\omega_i}$, and F_t is the Fresnel transmittance term. If both L_i and F_t are projected into

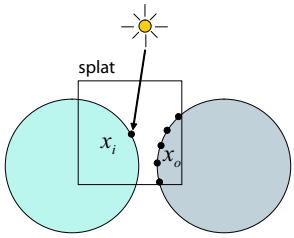

Fig. 6. Since our algorithm operates in image-space, a splat centered at a point x_i on one object can scatter light to recieving points x_o on the other object. The image on the right shows the bleeding of scattered light from one object to the other. This issue is resolved by using IDs to make sure that both the shooting and recieving points belong to the same object.

spherical harmonic basis, then the above integral reduces to a simple dot product operation of their coefficient vectors. The functions L_i and F_t are projected into SH coefficients which are then pre-rotated for a number of different coordinate frame orientations and stored in environment maps. Therefore for each direction vector $\overrightarrow{\omega}$ corresponding to a pixel in the environment map, the SH coefficients are rotated such that $\overrightarrow{\omega}$ forms the y-axis of the local coordinate frame (see Figure 5(a)). This enables us to look up the rotated SH coefficients from the environment maps using the normal vector at a particular point. After performing the dot product we obtain the incident light that can then be used in Render Pass 3 of the algorithm. It is important to note that in doing so the angular distribution of the transmitted light is lost. Consequently, it becomes difficult to accommodate single scattering in the existing framework without any pre-computation. However, multiple scattering can still be computed since the dipole diffuse reflectance function, R_d, depends only on the distance between two points.

The second issue with environment lighting is that of determining the points x_i that receive light. This is handled using depth peeling, alternating between the front and back faces at each layer. Figure 5(b) shows two such layers. The object is rendered multiple times, and in each render pass the depth values from the Z-buffer of the previous pass are used to discard the points that were rendered in that pass. The process is stopped when no points are rendered. Note that this results in a set of textures containing sample points x_i instead of just a single texture in Render Pass 1 of the algorithm (see Section 4.1). Therefore, the number of splats required is $n \times n \times l$, where l is the number of layers.

4.3 Rendering Multiple Objects

Our method is an image-space approach to computing subsurface scattering; that is, instead of directly working with the 3D geometry, our algorithm operates on 2D images of the geometry. Therefore other than the cost of the rasterization step in which the images of the geometry are constructed, there is no additional cost in processing complex geometric objects or scenes. Hence rendering a complex 3D object such as the Happy Buddha with subsurface scattering using our algorithm requires nearly the same amount of time as a simple object such as a sphere. Moreover, we are able to render multiple translucent objects with different scattering properties simultaneously for no additional cost. The necessary algorithm

Fig. 7. From left to right, objects with increasing geometric complexity rendered with subsurface scattering using our algorithm at rates of 30fps, 26fps, and 24fps respectively. Notice the geometric complexity does not greatly influence the render time due to our image-space approach.

modifications required for this are explained next.

The first modification involves the dipole texture. Recall that the 1-dimensional dipole diffuse reflectance function, R_d, is stored as a lookup table indexed by the distance. Notice that for each object that we want to render, a separate R_d lookup table must be created. We define a new function that accounts for multiple objects: R_d^ξ, where ξ is the object ID. For given set of objects, ξ is simply the index that identifies a given material and retrieves its absorption and scattering coefficients used in the computation of R_d. R_d^ξ is therefore computed and stored in a 2D texture where each row corresponds to the function R_d computed for the object ξ.

During render Pass 3 of the algorithm, the correct row of the dipole texture must be accessed in order to compute the multiple scattering term corresponding to the object's material. However, since the scattering is performed on a random set of points, x_i, rather than on a per object basis, the points are not ordered by material groups. Therefore in order to identify its material, an ID is stored as a vertex attribute for each x_i. Another problem with this method is that if two objects are placed close enough, they will contribute light due to multiple scattering to each other as shown in Figure 6. This issue is remedied by storing an object ID along with the 3D world positions for the points x_o in render Pass 2. Then a simple ID check is performed during the multiple scattering computation. If the object IDs of x_i and x_o are the same, multiple scattering computation proceeds as normal. If they are different, the computation is discarded by clipping the pixel.

5. RESULTS

We implemented our algorithm using HLSL and the Microsoft DirectX 9 SDK. The results presented in this section are from our implementation running on a 2.99 GHz Intel Pentium 4 PC with 1 GB of physical memory and a GeForce 7800 GPU. However, since our algorithm performs no computation on the CPU and does not use any of the system's main memory, a much lower end system would be sufficient. A shader model 3.0 compatible graphics card is required because our algorithm performs texture lookups in the vertex shader.

The major difference that separates our algorithm from most other rendering methods is that it is an image-space technique. Therefore the frame-rate obtained using our method is nearly the same regardless of the geometric complexity of the

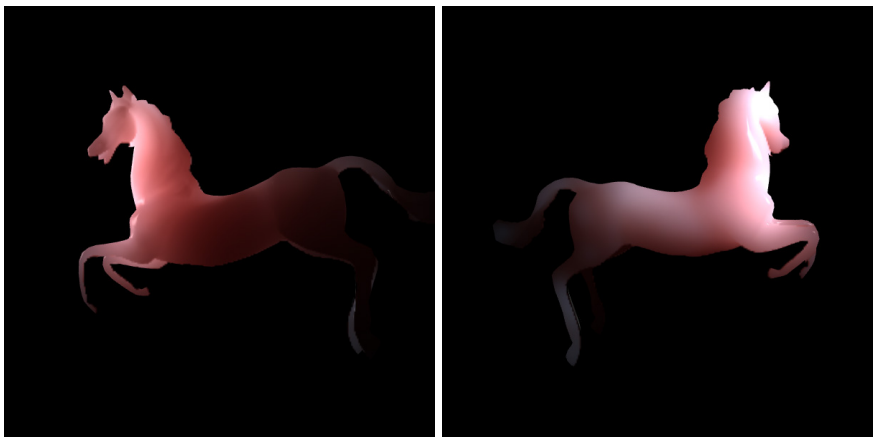

Fig. 8. Subsurface scattering in a back (left) and front (right) lit horse model. These images were rendered at 24fps at a resolution of 800×600 pixels.

object being rendered. The only dependence of the speed is on the resolution of the images being rendered: i.e. the resolution of the render target for obtaining the splat locations x_i, and the resolution of the scatter texture. On an average, with a scatter texture resolution of 800×600 pixels and 64×64 splats, we are able to obtain rates of around 25 frames per second. Figure 7 shows a sequence of images of objects with increasing level of geometric complexity rendered using our algorithm. Notice that the frame-rates are consistent, regardless of the object geometry. The small differences in the frame-rates are due to the rasterization overhead in the intermediate render passes of the algorithm.

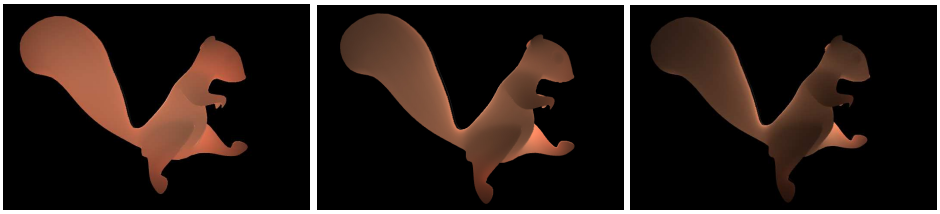

Fig. 9. Animation of a deformable squirrel model that is scaled down and up (left and right) dynamically (the images have been adjusted to maintain a consistent size of the model). The left image appears brighter since the scattered light has to travel a shorter distance compared to the right image of the scaled up squirrel model in which it has to travel a longer distance. Note how the light attenuation accentuates the features of the model.

Figure 8 demonstrates our algorithm's ability to render subsurface scattering in back lit objects. The increase in attenuation of the scattered light in the stomach and thigh regions of the horse gives the perception of depth, which is characteristic of back lit translucent objects. Since our algorithm has no pre-computation and is evaluated fully at every frame, it gives us the freedom to dynamically change lighting, viewing, and geometry. Figure 9 shows frames from an animation of

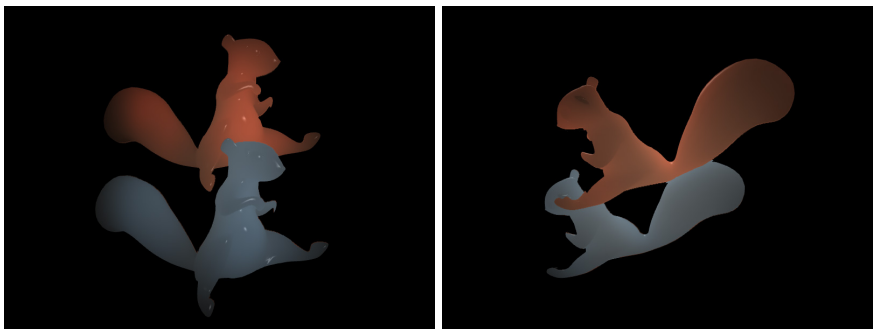

Fig. 10. Multiple objects with different scattering properties rendered at 25fps. Note that due to our image-space clipping technique, scattering from one object to the other is prevented even though the objects are almost adjoined.

subsurface scattering in a deformable object. In this experiment, the 3D model was nonuniformly enlarged and contracted to show the effect of change in geometry on the scattering behaviour. Another interesting feature of our image-space approach is that it allows us to render multiple objects with different scattering properties simultaneously and for no additional cost. Figure 10 shows images of two squirrel models with different absorption and scattering coefficients rendered at 25 fps. It also demonstrates the clipping technique explained in Section 4.3 to prevent scattering from one object to another. Figure 13 shows results from our algorithm with environment lighting.

Donner and Jensen recently introduced a multipole diffusion approximation for computing scattering in layered materials [Donner and Jensen 2005]. This new multipole function can be replaced with the dipole function in order to render objects made up of layers with different scattering properties. We demonstrate results using the multipole method in our rendering system in Figure 14.

5.1 Analysis

As shown in Figure 11, our algorithm produces results comparable to those obtained using offline renderers. The main difference between our method and other existing techniques that employ the dipole BSSRDF model is that we compute the area integral for multiple scattering in image-space. This is done by rendering quads at points x_i on the surface of the translucent object that is exposed to the light source. The quads provide access to points x_o visible to the camera, to which the light is scattered from each x_i. As is the case with all other methods, the sampling rate for x_i and x_o must be sufficient enough to prevent any visual artifacts. In our algorithm, the number of x_o points is determined by the resolution of the image being rendered, where as the number of x_i points is chosen by the user. Since the number of points x_i corresponds to the number of quads rendered, it directly influences the rendering speed. Choosing too few x_i will result in patchy artifacts as shown in Figure 12, whereas an excess of x_i will increase the render time. There are two things worth noting: first, for any given frame, scattering is computed only for the visible points x_o; and second, the geometric complexity of the scene does not influence the render time.

Fig. 11. From left to right: bird model rendered (a) using Lambertian diffuse and Phong specular lighting, (b) our subsurface scattering algorithm, and (c) an offline renderer. Note the translucency at the cheek and feet regions in the latter images.

As a direct consequence of the sampling strategy used to select the points x_i, temporal artifacts are introduced in certain cases. Recall that x_i correspond to the pixels of the render target texture on which the 3D world positions of the object have been rendered from the light's view (see Section 4.1, render Pass 3). Pixels that fall on the surface of the object get converted into sample points x_i, and the blank pixels are discarded. Therefore, the number of points x_i is in fact dependent on the orientation and positioning of the object as observed from the light's view. Thus, for example, when the object rotates, some flickering can be noticed due to the changing number of x_i. The amount of flickering also depends on the shape of the object being rendered. An object with fairly skewed dimensions is more likely to cause temporal artifacts as opposed to a symmetric object, such as a sphere, in which the artifacts are negligible. The appearance of these artifacts can be reduced in a number of different ways. One simple method is to restrict the extent of the image plane of the render target texture to the bounding box of the object in order to minimize the number of blank pixels. Another way is to dynamically increase or decrease the sampling rate depending on the number of non-blank pixels, which can be obtained using occlusion query. Therefore, if majority of the pixels fail the z-test, indicating that most of the pixels are blank, then the sampling rate for x_i should be increased, and vice versa. The aim is to maintain a constant number of sample points x_i at each frame. We employ the former technique in our implementation.

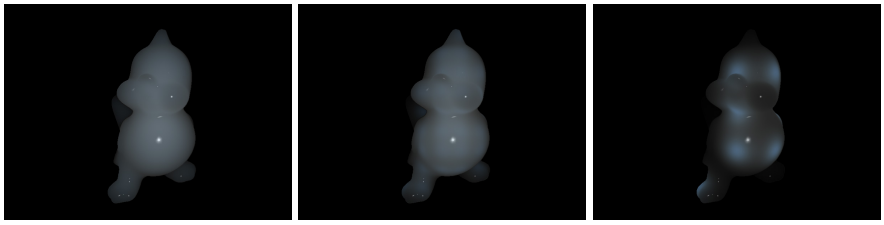

Fig. 12. From left to right: the bird model rendered using 32×32 samples at 31 fps, 16×16 samples at 65 fps, and 8×8 samples at 151 fps. Undersampling results in "patchy" artifacts because the splats do not overlap sufficiently.

6. CONCLUSION AND FUTURE WORK

We have presented a real-time image-space algorithm for rendering translucent objects with subsurface scattering using the dipole BSSRDF model, accounting for both the single and multiple scattering terms. We compute the area integral using a shooting approach in contrast to the traditional gathering approach used by existing methods, which allows us to implement the algorithm entirely on the GPU. We employ a dual camera-light view representation for storing the incident light and performing the scattering operation, which enables us to account for all the subsurface scattering effects as well avoid artifacts associated with space conversion. Furthermore this approach is also efficient since it only performs the scattering computation between significant point pairs, thus eliminating any unnecessary calculations. Our method is able to produce visually convincing results at rates of about 25 fps for arbitrarily complex objects under dynamic view and illumination.

For future work we would like to accommodate translucent objects with heterogeneous subsurface scattering properties in our current real-time rendering system. We would also like to explore GPU based sampling strategies for selecting x_i that would help us further reduce temporal artifacts.

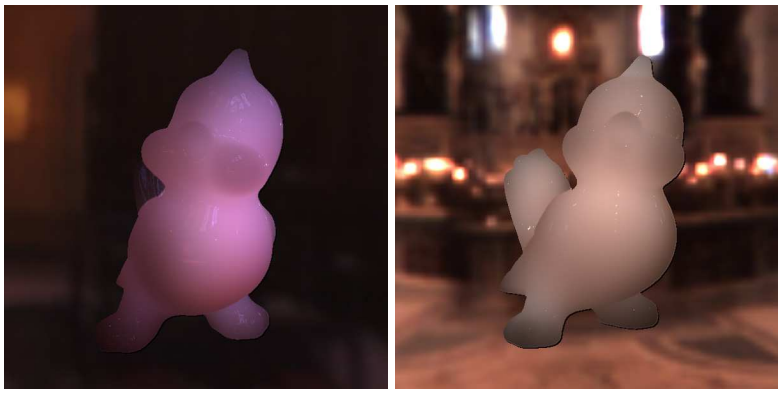

Fig. 13. The bird model rendered using our algorithm under environment lighting: (left) Grace Cathedral and (right) St. Peters. Note that only the multiple scattering term of the BSSRDF model is used.

REFERENCES

Carr, N. A., Hall, J. D., and Hart, J. C. 2003. GPU algorithms for radiosity and subsurface scattering. In *HWWS '03: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*. Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 51–59.

Dachsbacher, C. and Stamminger, M. 2003. Translucent shadow maps. In *EGRW '03: Proceedings of the 14th Eurographics workshop on Rendering*. Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 197–201.

Donner, C. and Jensen, H. W. 2005. Light diffusion in multi-layered translucent materials. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Papers*. ACM Press, New York, NY, USA, 1032–1039.

Fig. 14. Multilayered material rendered using the multipole diffusion method with our algorithm. The material is made of a thin yellow layer and a green layer. The second and third images show the effect of increasing and decreasing the thickness of the yellow layer.

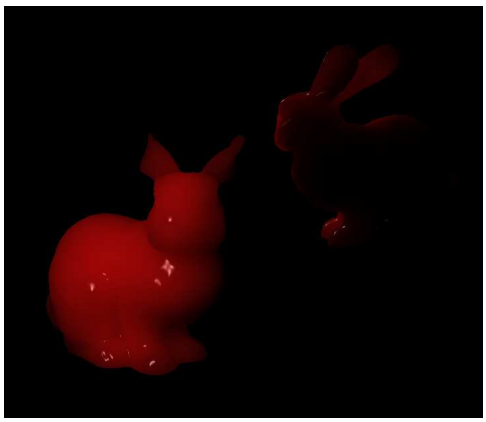

Fig. 15. The Stanford Bunny model made of a highly absorbing material. Notice how the back lighting effect at the sillhouettes rapidly decreases toward the center of the model due to the high material density.

Francois, G., Pattanaik, S., Bouatouch, K., and Breton, G. 2006. Subsurface texture mapping. *IEEE Computer Graphics and Applications (to appear)*.

Hao, X. and Varshney, A. 2004. Real-time rendering of translucent meshes. *ACM Trans. Graph. 23,* 2, 120–142.

Jensen, H. W. and Buhler, J. 2002. A rapid hierarchical rendering technique for translucent materials. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*. ACM Press, New York, NY, USA, 576–581.

Jensen, H. W., Marschner, S. R., Levoy, M., and Hanrahan, P. 2001. A practical model for subsurface light transport. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*. ACM Press, New York, NY, USA, 511–518.

Lensch, H., Goesele, M., Bekaert, P., Kautz, J., Magnor, M., Lang, J., and Seidel, H. 2002. Interactive rendering of translucent objects. In *H. P. A. Lensch, M. Goesele, P. Bekaert, J. Kautz, M. A. Magnor, J. Lang, and H.-P. Seidel. Interactive rendering of translucent objects. In Proc. Pacific Graphics 2002, pages 214–224, Oct. 2002.*

Mertens, T., Kautz, J., Bekaert, P., Reeth, F. V., and Seidel, H.-P. 2003. Efficient rendering of local subsurface scattering. In *PG '03: Proceedings of the 11th Pacific Conference on Computer Graphics and Applications*. IEEE Computer Society, Washington, DC, USA, 51.

Mertens, T., Kautz, J., Bekaert, P., Seidelz, H.-P., and Reeth, F. V. 2003. Interactive rendering of translucent deformable objects. In *EGRW '03: Proceedings of the 14th Eurographics workshop on Rendering*. Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 130–140.

Ng, R., Ramamoorthi, R., and Hanrahan, P. 2003. All-frequency shadows using non-linear wavelet lighting approximation. *ACM Trans. Graph. 22,* 3, 376–381.

Shah, M., Konttinen, J., and Pattanaik, S. 2006. Caustics mapping: An image-space technique for real-time caustics. *IEEE Transactions on Visualization and Computer Graphics (to appear)*.

Sloan, P.-P., Hall, J., Hart, J., and Snyder, J. 2003. Clustered principal components for precomputed radiance transfer. *ACM Trans. Graph. 22,* 3, 382–391.

Sloan, P.-P., Luna, B., and Snyder, J. 2005. Local, deformable precomputed radiance transfer. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Papers*. ACM Press, New York, NY, USA, 1216–1224.

Szirmay-Kalos, L., Aszódi, B., Lazányi, I., and Premecz, M. 2005. Approximate ray-tracing on the GPU with distance impostors. In *Proceedings of Eurographics 2005*.

Wang, R., Tran, J., and Luebke, D. 2005. All-frequency interactive relighting of translucent objects with single and multiple scattering. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Papers*. ACM Press, New York, NY, USA, 1202–1207.

Wyman, C. 2005. An approximate image-space approach for interactive refraction. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Papers*. ACM Press, New York, NY, USA, 1050–1053.

Wyman, C. and Davis, S. 2006. Interactive image-space techniques for approximating caustics. In *SI3D '06: Proceedings of the 2006 symposium on Interactive 3D graphics and games*. ACM Press, New York, NY, USA, 153–160.