

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

Musawir A. Shah, Jaakko Konttinen, and Sumanta Pattanaik ■ *University of Central Florida*

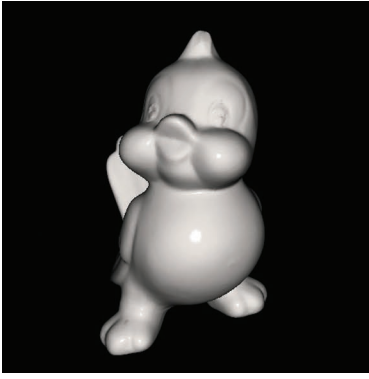
The authors present an algorithm for real-time realistic rendering of translucent materials such as marble, wax, and milk. Their method captures subsurface scattering effects while maintaining interactive frame rates. The main idea of this work is that it employs the dipole diffusion model with a splatting approach to evaluate the integral over the surface area for computing illumination due to multiple scattering.

Most real-world materials exhibit translucency at an appropriate scale. The subsurface interaction of light involves light entering through the surface and scattering multiple times inside the material, ultimately exiting the surface in a diffuse manner. For materials with relatively low absorption but high scattering property, such as marble, the light scatters around a larger distance under the surface and exits the surface at locations other than where it entered. The translucency's extent depends on the material's scattering and absorption property. Certain highly scattering materials give rise to unique optical phenomena, such as the appearance of a backlit object, which distinguishes their visual look from other opaque solids. An important goal in computer graphics is to faithfully represent these materials for realistic image synthesis. BRDFs (bidirectional reflectance distribution functions)

are generally used to model the reflectance properties of most opaque materials. This modeling is based on the assumption that light enters and exits from the same location on the surface. Because of this assumption, BRDFs aren't sufficient for translucent objects.

Instead, translucency is modeled using the BSSRDF (bidirectional surface scattering reflectance distribution function), which takes into account both the angular and spatial distribution of light on an object's surface. A BSSRDF is an 8D function that describes the light transport from one point to another for a given illumination and viewing direction.

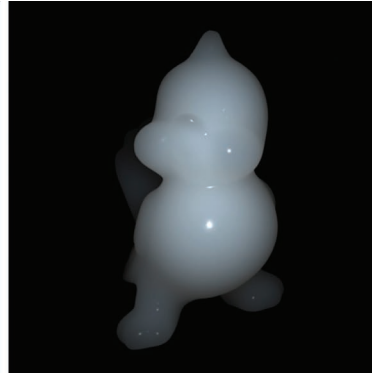
Researchers have used numerous offline rendering methods that simulate subsurface scattering to render translucent materials. Although these methods produce good-quality images, the computation is generally expensive. With the recent introduction of the dipole BSSRDF model that employs the dipole source approximation for multiple scattering,¹ several fast rendering algorithms have emerged. Our work, inspired by the dipole BSSRDF model, focuses 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 (see Figure 1). For other methods, see the "Subsurface Scattering Approaches" sidebar (page 68). Our main contribution is that we use a dual light-camera space technique to efficiently evaluate the subsurface scattering by only considering interaction between significant point pairs. The area integral is computed via a splatting approach in an image-space framework, thus allowing support for arbitrarily complex geometry. We show that our algorithm can be implemented entirely on the GPU and is easily integrated into existing real-



(a)



(b)



(c)

Figure 1. Bird model rendered using (a) Lambertian diffuse and Phong specular lighting, (b) our subsurface scattering algorithm (27 fps), and (c) an offline renderer (approximately 3 minutes per frame). Note the translucency at the cheek and feet regions in the latter images.

time rendering systems by inserting only three additional render passes.

Background

Our algorithm employs the BSSRDF model for computing the subsurface scattering in a given homogeneous object. BSSRDFs are one level higher than BRDFs in the hierarchy of models for describing object appearance. A BRDF is a 4D 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 exits at a different location. The BSSRDF is an 8D 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 derived as

$$dL_o(x_o, \omega_o) = S(x_i, \omega_i, x_o, \omega_o) d\Phi(x_i, \omega_i) \\ L_o(x_o, \omega_o) = \int_A \int_{2\pi} S(x_i, \omega_i, x_o, \omega_o) L_i(x_i, \omega_i) (\vec{n}_i \cdot \vec{\omega}_i) d\omega_i dA(x_i) \quad (1)$$

The various symbols used throughout this article are defined in Table 1. For highly scattering, optically thick materials, the BSSRDF, $S(x_i, \omega_i, x_o, \omega_o)$, is approximated as a sum of a single scattering term $S^{(1)}$ and a diffuse multiple scattering term S_d :

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

For highly scattering translucent materials it is sufficient to only consider the BSSRDF's diffuse multiple-scattering term. Most current interactive BSSRDF rendering techniques therefore don't account for the single-scattering term, which enables them to simplify computation and increase speed.

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 Henrik Wann Jensen and his colleagues¹ as follows:

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

R_d is defined as

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

where $\alpha' = \sigma'_s / \sigma'_t$ is the reduced albedo obtained from the reduced scattering and extinction coefficients, and $\sigma_{tr} = \sqrt{3\sigma_a \sigma'_t}$ 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 ,

Table 1. Symbols in the multiple scattering computation using the dipole source BSSRDF model.

Symbol	Description
ω_i	Incident light direction
ω_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'_s	Reduced extinction coefficient = $\sigma_a + \sigma'_s$
η	Relative refractive index
$F_t(\eta, \omega)$	Fresnel transmittance
$p(\omega_i, \omega_o)$	Phase function

Subsurface Scattering Approaches

Subsurface scattering in translucent materials has been studied extensively in computer graphics. Impressive results have been produced from offline rendering techniques that simulate light transport in participating media. Although these approaches faithfully reproduce most subsurface scattering effects, they are generally slow. Faster algorithms have emerged after the introduction of the work by Henrik Wann Jensen and his colleagues, which employs the dipole source approximation for multiple scattering in the bidirectional surface scattering reflectance distribution function (BSSRDF) model.¹ Soon after, Jensen and Juan Buhler presented a two-pass hierarchical approach that significantly increased the computation speed of the multiple scattering.² In this approach, irradiance samples are taken on the surface of the object and organized into an octree data structure. This allows for hierarchical integration, where the neighboring points are sampled densely and those further away are considered in groups. Recent methods have built on this technique and accelerated the multiple scattering computation further using graphics hardware. These algorithms closely resemble radiosity; the scattering event is treated as point-to-point,^{3,4} point-to-patch,⁵ and patch-to-patch⁶ form-factorlike transport. Although rendering at interactive frame rates has been shown using these techniques, they can't handle environmental lighting efficiently. Furthermore, to maintain interactivity, precomputation is required to establish the form-factor links in most cases. Unlike its counterparts, the technique presented by Tom Mertens and his colleagues can handle deformable geometry.⁵ They provide three different rendering modes, of which the on-the-fly mode offers the most flexibility because it involves full evaluation from scratch. However, it's also the slowest rendering mode, producing an average frame rate of about 9 fps.

Subsurface scattering was included in the precomputed radiance transfer (PRT) framework by Peter-Pike Sloan and his colleagues.⁷ They precompute and store the multiple scattering component of the BSSRDF as transport vectors on a truncated lower-order spherical-harmonics (SH) basis. Scattering SH coefficients are then combined with the SH coefficients of the environment lighting function at the render stage. Sloan, Ben Luna, and John Snyder added support for deformable objects to the PRT framework.⁸ Support for all-frequency lighting was first introduced for the case of diffuse BRDF rendering by Ren Ng and colleagues⁹ and was recently adapted to include full BSSRDF rendering accounting for both single and multiple scattering terms.¹⁰

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 the traditional convention of operating on geometry, we adopted an image-space approach, which allows us to efficiently employ graphics hardware for the scattering computation as well as support objects of arbitrary geometric complexity without compromising real-time frame rates.

Image-space algorithms operate on rasterized images of 3D scenes rather than geometry and are suitable for implementation on GPUs. 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.¹¹⁻¹⁴ Steps toward real-time global illumination are also being taken in the same spirit. Real-time indirect illumination for diffuse and nondiffuse surfaces can now be shown using an image-space splatting approach.¹⁵ Tom Mertens and his colleagues¹⁶ present an image-space subsurface scattering algorithm that uses the dipole source approximation. They precompute a set of sample points for the area integration and evaluate

respectively (see Figure 2a, page 70). d_r and d_v are distances from x_o to the real and virtual dipole lights. We compute these distances as follows:

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

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

$$r = ||x_o - x_i||$$

$$z_r = \frac{1}{\sigma_t}$$

$$z_v = z_r \left(1 + \frac{4A}{3}\right)$$

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

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

Substituting S_d in Equation 1 we obtain the outgoing radiance equation:

$$L_o(x_o, \vec{\omega}_o) = \frac{F_t(\eta, \vec{\omega}_o)}{\pi} \int_A R_d(||x_i - x_o||) E(x_i) dA(x_i), \quad (3)$$

where E is the transmitted irradiance defined as

$$E(x_i) = \int_{2\pi} L_i(x_i, \vec{\omega}_i) F_t(\eta, \vec{\omega}_i) (\vec{n}_i \cdot \vec{\omega}_i) d\omega_i.$$

For a point light source, the expression $E(x_i)$ is simpler:

$$E(x_i) = \frac{I(\vec{\omega}_i)}{d_{pl}^2} F_t(\eta, \vec{\omega}_i) (\vec{n}_i \cdot \vec{\omega}_i),$$

the integral using multiple GPU passes. Similarly, Carsten Dachsbacher and Marc Stamminger¹⁷ present an algorithm that employs translucent shadow maps, treating the integration computation as a filtering operation.

In this article, we present an image-space algorithm for real-time subsurface scattering in translucent materials. Our method also uses the dipole source approximation BSSRDF model and supports both point light and environment illumination. The algorithm doesn't require any precomputation and is evaluated fully at every frame. We obtain results comparable to those produced using offline renderers at about 25 fps for arbitrarily complex objects.

References

1. H.W. Jensen et al., "A Practical Model for Subsurface Light Transport," *Proc. Siggraph*, ACM Press, 2001, pp. 511–518.
2. H.W. Jensen and J. Buhler, "A Rapid Hierarchical Rendering Technique for Translucent Materials," *Proc. Siggraph*, ACM Press, 2002, pp. 576–581.
3. H. Lensch et al., "Interactive Rendering of Translucent Objects," *Proc. Pacific Graphics*, IEEE CS Press, 2002, pp. 214–224.
4. X. Hao and A. Varshney, "Real-Time Rendering of Translucent Meshes," *ACM Trans. Graphics*, vol. 23, no. 2, 2004, pp. 120–142.
5. T. Mertens et al., "Interactive Rendering of Translucent Deformable Objects," *Proc. 14th Eurographics Workshop Rendering (EGRW 03)*, Eurographics Assoc., 2003, pp. 130–140.
6. N.A. Carr, J.D. Hall, and J.C. Hart, "GPU Algorithms for Radiosity and Subsurface Scattering," *Proc. ACM Siggraph/Eurographics Conf. Graphics Hardware (HWWG 03)*, Eurographics Assoc., 2003, pp. 51–59.
7. P.-P. Sloan et al., "Clustered Principal Components for Precomputed Radiance Transfer," *ACM Trans. Graphics*, vol. 22, no. 3, 2003, pp. 382–391.
8. P.-P. Sloan, B. Luna, and J. Snyder, "Local, Deformable Precomputed Radiance Transfer," *ACM Trans. Graphics (Proc. Siggraph)*, vol. 24, no. 3, 2005, pp. 1216–1224.
9. R. Ng, R. Ramamoorthi, and P. Hanrahan, "All Frequency Shadows Using Non-linear Wavelet Lighting Approximation," *ACM Trans. Graphics (Proc. Siggraph)*, vol. 22, no. 3, 2003, pp. 376–381.
10. R. Wang, J. Tran, and D. Luebke, "All-Frequency Interactive Relighting of Translucent Objects with Single and Multiple Scattering," *Proc. Siggraph*, ACM Press, 2005, pp. 1202–1207.
11. L. Szirmay-Kalos et al., "Approximate Ray-Tracing on the GPU with Distance Impostors," *Proc. Eurographics*, Blackwell Publishing, 2005.
12. M. Shah, J. Konttinen, and S. Pattanaik, "Caustics Mapping: An Image-Space Technique for Real-Time Caustics," *IEEE Trans. Visualization and Computer Graphics*, vol. 13, no. 2, 2007, pp. 272–280.
13. C. Wyman, "An Approximate Image-Space Approach for Interactive Refraction," *Proc. Siggraph*, ACM Press, 2005, pp. 1050–1053.
14. C. Wyman and S. Davis, "Interactive Image-Space Techniques for Approximating Caustics," *Proc. 2006 Symp. Interactive 3D Graphics and Games (SI3D 06)*, ACM Press, 2006, pp. 153–160.
15. C. Dachsbacher and M. Stamminger, "Splatting Indirect Illumination," *Proc. 2006 Symp. Interactive 3D Graphics and Games (SI3D 06)*, ACM Press, 2006, pp. 93–100.
16. T. Mertens et al., "Efficient Rendering of Local Subsurface Scattering," *Proc. 11th Pacific Conf. Computer Graphics and Applications (PG 03)*, IEEE CS Press, 2003, p. 51.
17. C. Dachsbacher and M. Stamminger, "Translucent Shadow Maps," *Proc. 14th Eurographics Workshop Rendering (EGRW 03)*, Eurographics Assoc., 2003, pp. 197–201.

where d_{pl} is the distance from the point light source to x_i . 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 object surface. Thus the cost of the final rendering is quadratic in the number of sample points (sample points seen by the light \times sample points seen by the camera), which can be quite expensive. Most recent work has focused on finding ways for performing this integration process efficiently by using hierarchical data structures and precomputation.

Our Algorithm

We compute the area integration in Equation 3 by a splatting process rather than a gathering process. We take sample points on the surface visible from the light source and splat the scattering contribution to all points visible to the viewer within the effective scattering range from each point. Each point on the surface rendered receives the scattering contribution from all points that influence it.

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

Web Extra

To view a video of our algorithm, visit <http://computingnow.computer.org/cgavideos>. The video shows an animation of subsurface scattering in a deformable object. In this experiment, the 3D teapot model is dynamically displaced to show the effect of geometry change on scattering behavior.

that the light's view captures all points that receive light. However, light can be scattered to other points on the object's surface that aren't 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 scattering in backlit objects will not be attained. On the other hand, if the scatter texture is stored in the camera's view space, scattering to all visible points will be computed. Although some points that could potentially receive light due to scattering are still left out, they're insignificant because they aren't visible to the viewer. Furthermore, other than accurately accounting for the scattering, this dual-space representation also helps eliminate stretching artifacts at grazing angles due to undersampling that occurs when points are projected from the camera's view space to the light's view as 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 because no conversion from one space to the other is involved.

Our splatting approach has two main advantages over the traditional integration approach of gathering scattering contribution from nearby points. First, we only compute scattering between significant point pairs—that is, pairs of splatting and gathering points, so that the splatting points must have incident light to scatter and the receiving points must be visible to the viewer. Second, treating the integration computation in this fashion allows for a simple and natural implementation on the GPU using additive alpha blending.

Algorithm Implementation Steps

Our algorithm runs entirely on the GPU. Therefore, it's 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.

In pass 1, the algorithm renders the object from the light's view. The pixel shader outputs 3D world space positions and normals to a texture. This gives a set of splatting points x_i .

During pass 2, the algorithm renders the object from the camera's view. The pixel shader outputs 3D world positions to a texture. This texture is the set of x_o points.

In pass 3, the algorithm renders $n \times n$ screen-aligned quads (or splats) from the camera's view. This creates 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 supported by shader model 3.0 and above. Alternatively, we can use OpenGL's render-to-vertex buffer extension to output splat positions. In the pixel shader, the algorithm computes the multiple scattering term at each pixel x_p . Additive alpha blending is enabled for this pass to accumulate the scattering contribution from each splat.

In the final render pass, the algorithm renders the object with subsurface scattering using the scatter texture and adds the specular reflections. Note that the algorithm has two variable quantities: the total number of splats used and the size of individual splats. We first compute the splat size from the scattering properties of the material being rendered and then calculate the total number of splats required. We compute the splat size by determining the maximum extent of the spatial domain of integration, r_{\max} , so that the integral of the dipole diffusion function over the truncated area $A' = r_{\max}^2$ is within some threshold fraction of the integral over the entire surface area, A :

$$\frac{\int_A R_d(x) dA(x) - \int_{A'} R_d(x) dA'(x)}{\int_A R_d(x) dA(x)} < \varepsilon. \quad (4)$$

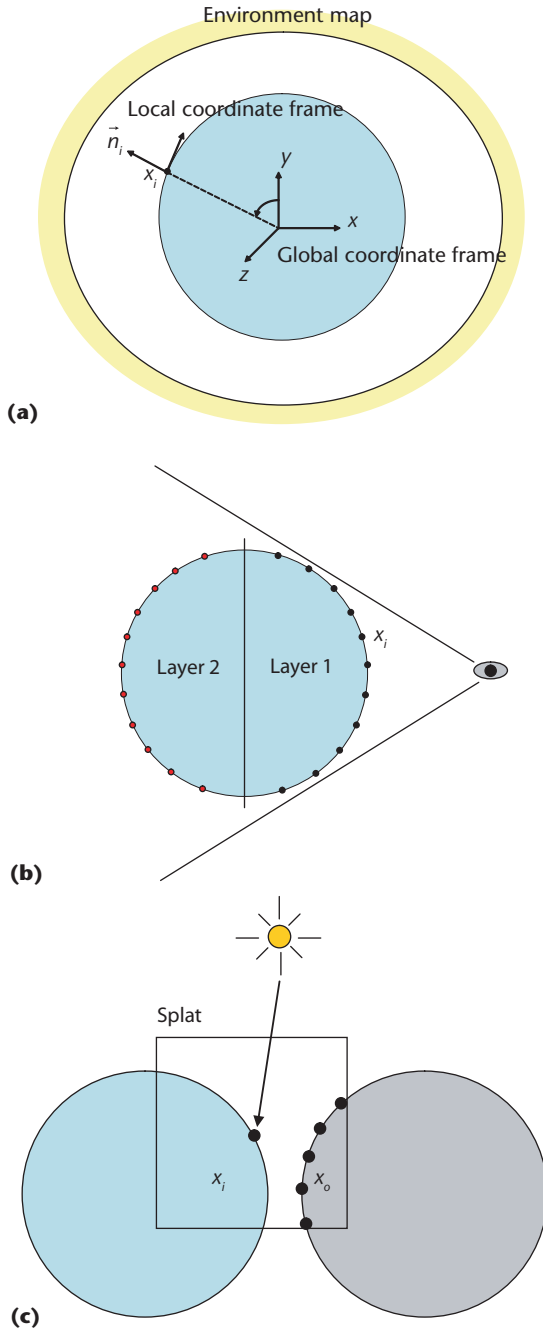
We numerically compute the value of r_{\max} that satisfies this inequality. r_{\max} is the splat size and A' is the effective scattering range. Next, the dipole function and r_{\max} determine the number of overlapping splats, n_o , required at each point to compute the integral in Equation 3. We employ Weber's law to obtain the number of splats required so that adding more splats will not result in a perceptible change in the final value:

$$\frac{R_d(0)}{n_o \times R_d(r_{\max})} < \varepsilon. \quad (5)$$

We can easily compute n_o from this equation. The user can adjust the error thresholds, ε , in Equations 4 and 5 to obtain the desired rendering quality and

Figure 4.

(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 . (c) Because our algorithm operates in image-space, a splat centered at a point x_i on one object can scatter light to receiving points x_o on the other object.



speed. The results in this article were created using $\varepsilon = 0.01$.

Finally, on the basis of the number of overlapping splats and the size of individual splats, we can compute the total number of splats required in our uniform sampling scheme. If the width of the light's view frustum in world units is W , then the number of splats required is $n \times n$, where

$$n = \frac{\frac{1}{2} n_o W}{r_{\max}}.$$

Note the inverse relationship between the number of splats and the size of each splat. Therefore, for materials with a large mean-free path length

and hence a large splat size, the number of splats will be small. On the other hand, if the mean-free path length is small, the splat size will also be small but the number of splats will be large. Because our algorithm is fill-rate dependent and the total number of fragments processed is given by the number of splats times the splat size, the average rendering time for all types of materials with small or large mean-free path lengths is the same.

Environment Lighting

So far we have discussed our subsurface scattering algorithm in the context of point light illumination. However, we can easily extend the algorithm to support environment lighting. There are two key differences between point light illumination and environment lighting. First, the incident light at each point on the surface of the object has a hemispherical distribution instead of a single illumination direction. Second, all points on the surface can potentially receive light, as opposed to the point light illumination case, in which only the points visible from the light's view can receive light. We manage the first issue with spherical harmonics (SH) to represent the environment lighting.

To compute the transmitted light at a given point x_i from the environment, we must evaluate the following integral:

$$E(x_i) \int_{2\pi} L_i(\vec{\omega}_i) F_t(\eta, \vec{\omega}_i) (\vec{n}_i \cdot \vec{\omega}_i) d\vec{\omega}_i,$$

where $L_i(\vec{\omega}_i)$ is the value of the environment lighting function in direction $\vec{\omega}_i$, and F_t is the Fresnel transmittance term. Combining the cosine term and the Fresnel transmittance function we obtain

$$E(x_i) = \int_{2\pi} L_i(\vec{\omega}_i) F_t' d\vec{\omega}_i$$

$$F_t' = F_t(\eta, \vec{\omega}_i) (\vec{n}_i \cdot \vec{\omega}_i).$$

If both L_i and F_t' are projected into the SH basis, then the previously mentioned 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 prerotated for several different coordinate frame orientations and stored in environment maps. Therefore, for each direction vector $\vec{\omega}$ corresponding to a pixel in the environment map, the SH coefficients are rotated so that $\vec{\omega}$ forms the y-axis of the local coordinate frame (see Figure 4a). 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. Note that in doing the dot product the transmitted light's angular distribution is lost. However, multiple scattering can still be computed because 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 with depth peeling,² alternating between the front and back faces at each layer. Figure 4b 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 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. Therefore, the number of splats required is $n \times n \times l$, where l is the number of layers.

Rendering Multiple Objects

We can render multiple translucent objects with different scattering properties simultaneously at no additional cost. Here we explain the necessary algorithm modifications required for this.

The first modification involves the dipole texture. Recall that the 1D dipole diffuse reflectance function, R_d , is stored as a lookup table indexed by the distance. Notice that for each object 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 a given set of objects, ξ is simply the index that identifies a given material and retrieves its absorption and scattering coefficients 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 .

The second modification involves storing the object ID as the object's vertex attribute to access the correct row of the dipole texture during the splatting step. This is required because the scattering is performed at a random set of points, x_i , rather than on a per-object basis.

If two objects are placed close enough, some splats centered at points on one object might cover the pixels occupied by the second object, as Figure 4c shows. In such a situation the two objects will contribute light due to multiple scattering to each other. We can remedy this issue by storing an object ID along with the 3D world positions for the points x_o in render pass 2. Then a simple ID

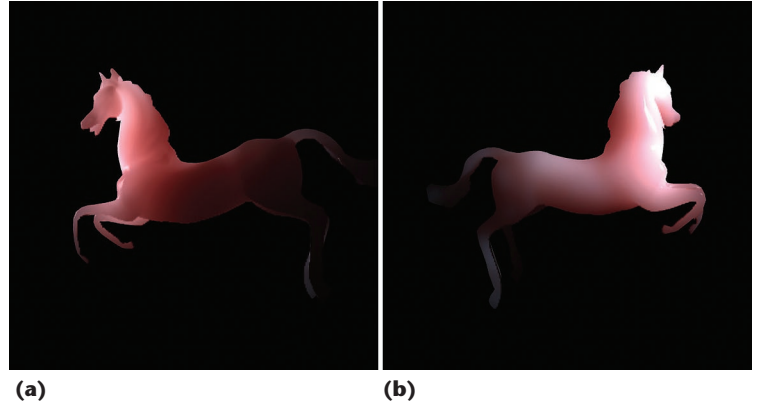


Figure 5. Subsurface scattering in a (a) back-lit and (b) front-lit horse model. Unlike certain existing methods, our algorithm places no restrictions on the orientation of the light source. These images were rendered at 24 fps at a resolution of 800×600 pixels.

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're different, the algorithm discards the computation by clipping the pixel.

For a brief discussion of our algorithm and other techniques, see the "Comparison of Other Work" sidebar (next page).

Results

We implemented our algorithm using High-Level Shading Language (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 Gbyte of physical memory and a GeForce 7800 GPU. However, because our algorithm performs no computation on the CPU and doesn't use any of the system's main memory, a much lower-end PC would suffice. A shader model 3.0 compatible graphics card is required because our algorithm performs texture lookups in the vertex shader.

Because our algorithm operates in image-space, the frame rate obtained with our method is virtually the same for any object we render regardless of its geometric complexity. The only dependence of the speed is on the size of individual splats, the total number of splats, and the resolution of the scatter texture. The number of splats and size of each splat are computed on the basis of the material properties of the object rendered. In our experiments using an error threshold of 0.01 in Equations 4 and 5, the total number of splats ranged from 32×32 for a large mean-free path lengths (see Figure 5) to 300×300 for small mean-free path lengths (see Figure 6, next page).

To analyze the impact of the mean-free path length on our algorithm's performance, we rendered the Happy Buddha model with three different

Comparison of Other Work

The key difference between the technique by Tom Mertens and his colleagues and our algorithm is that we adopt a splatting approach to evaluating the area integral in which the scattered light is shot from the light receiving points to points visible from the camera.^{1,2} By considering only these significant point pairs, the integration computation is performed more efficiently as opposed to summing up scattering contributions from all the neighboring points. Because our splatting approach doesn't involve any summation loops, it allows for a more natural implementation on the GPU, taking advantage of hardware blending for the accumulation. Furthermore, scattering in backlit objects in the method proposed by Mertens and his colleagues is hindered by the representation of the irradiance. They store the irradiance in a texture from the camera's view, in which case the backlit scattering effect cannot be achieved. Alternatively, the irradiance texture can be stored from the light's view to obtain backlit scattering; however, this results in artifacts due to stretching when the camera and light views are perpendicular.

In a recent update to their algorithm, Mertens and his colleagues proposed storing the irradiance in texture space.² This allows for backlit scattering because the irradiance field is defined everywhere on the object's surface. However, as the authors point out, a bijective and continuous parameterization of the object's surface must be available to store the irradiance in texture space. Furthermore, this technique is restricted to nondeformable objects. In our algorithm we use a dual light-camera view approach that enables us to capture backlit scattering in deformable objects and eliminate stretching artifacts.

Carsten Dachsbacher and Marc Stamminger present a

subsurface scattering algorithm that employs translucent shadow maps.³ 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. To speed up the rendering, they approximate the computation by separating the scattering into local and global responses. At the global level, a heuristically determined filtering pattern uses the depth and spatial variation between two points to compute the dipole reflectance, whereas at the local level the depth variation is ignored. In our algorithm, the distance between the scattering point, x_i , and the visible point, x_o , is computed explicitly, and therefore the dipole reflectance function is queried accurately. Because Dachsbacher and Stamminger don't provide comparisons to reference rendering results, it's difficult to verify the accuracy of the images in their work. Furthermore, owing to GPU storage limitations they are only able to support directional lighting, whereas our approach manages point light illumination as well as environment lighting.

References

1. T. Mertens et al., "Efficient Rendering of Local Subsurface Scattering," *Proc. 11th Pacific Conf. Computer Graphics and Applications* (PG 03), IEEE CS Press, 2003, p. 51.
2. T. Mertens et al., "Efficient Rendering of Local Subsurface Scattering," *Computer Graphics Forum*, vol. 24, no. 1, 2005, pp. 41–49.
3. C. Dachsbacher and M. Stamminger, "Translucent Shadow Maps," *Proc. 14th Eurographics Workshop Rendering* (EGRW 03), Eurographics Assoc., 2003, pp. 197–201.

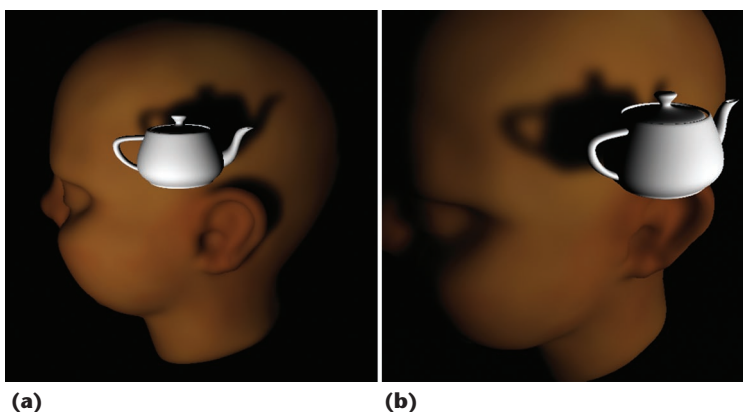


Figure 6. (a) Shadows cast from external and self-occlusion. (b) Close-up of the blurred shadow due to scattering. This figure demonstrates the algorithm's ability to correctly manage shadows on translucent objects.

levels of translucency. Figure 7 shows the results. As the extinction coefficient increases, the number of splats required increases but the size of each

individual splat decreases. Note that although the frame rate decreases slightly with the increased number of splats, it remains more or less constant because the decrease in the size of the splats compensates for the extra splats. Figure 8 shows the relationship of the mean-free path length with the total frame rate and the time taken to render individual splats.

On average, with a scatter texture resolution of 800×600 pixels we can obtain rates of around 25 fps. Figure 9 shows a sequence of objects with increasing levels of geometric complexity rendered with our algorithm. Notice that the frame rates are consistent regardless of the object geometry. Small differences in the frame rates are due to the rasterization overhead in the algorithm's intermediate render passes.

Figure 5 demonstrates our algorithm's ability to render subsurface scattering in backlit objects. The increase in attenuation of the scattered light in

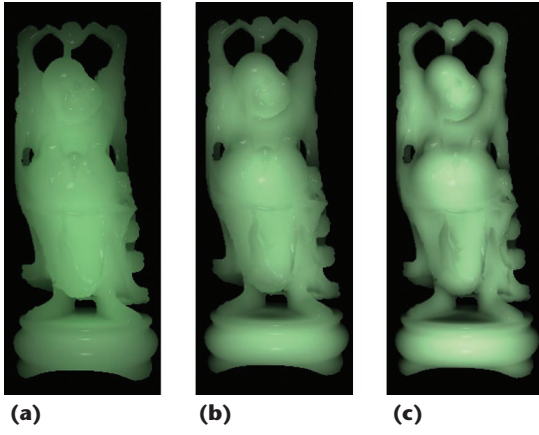


Image	Frames per second	Extinction coefficient (σ_t)	Number of samples	Sample size
(a)	33	0.27	51×51	0.63
(b)	27	1.47	71×71	0.24
(c)	24	3.27	114×214	0.16

Figure 7. Rendering performance statistics for translucent materials with varying mean-free path lengths.

the stomach and thigh regions of the horse gives the perception of depth, which is characteristic of backlit translucent objects. Because our algorithm has no precomputation and is evaluated fully at every frame, it gives us the freedom to dynamically change lighting, viewing, and geometry. The supplementary video (visit <http://computingnow.computer.org/cgavideos>) shows an animation of subsurface scattering in a deformable object. In this experiment, the 3D teapot model is dynamically displaced to show the effect of geometry change on scattering behavior. Another interesting feature of our approach is that it lets us render multiple objects with different scattering properties simultaneously at no additional cost. Figure 10 (next page) shows images of two squirrel models with different absorption and scattering coefficients rendered at 25 fps. It also demonstrates the clipping technique to prevent scattering from one object to another. Figures 10c and 10d (next page) show results from our algorithm with environment lighting.

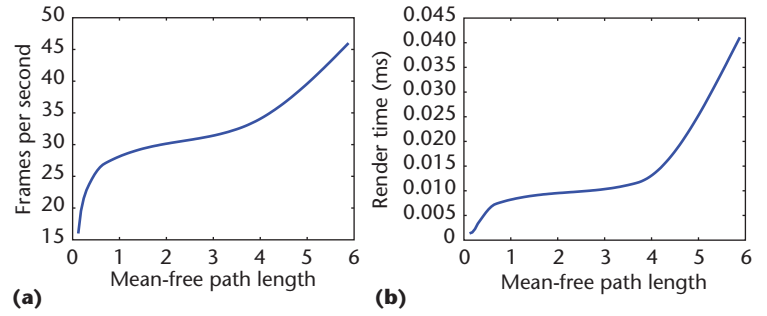


Figure 8. (a) Graph showing the relationship between the mean-free path length and the rendering frame rate of the algorithm. As the mean-free path length gets larger, the number of splats decreases and therefore the frame rate increases. (b) Graph showing the relationship between the mean-free path length and the time taken to render individual splats. As the mean-free path length gets larger, the splat size increases and therefore takes more time to render.

Our algorithm effectively manages shadowing from other occluding objects as well as self-shadowing. Figure 6 shows images of a human-head model rendered using our algorithm. Note the blurring of the shadows cast by the teapot and the ear owing to subsurface scattering. Self-shadowing is obtained without any additional steps, whereas shadows from other objects require a shadow map lookup to determine irradiance at splat locations, x_i . The images in Figure 6 were rendered at 23 fps. A video clip of this rendering is included in the supplementary video.

Craig Donner and Henrik Wann Jensen³ recently introduced a multipole diffusion approximation for computing scattering in layered materials. This new multipole function can substitute for the dipole function to render objects made up of layers with different scattering properties. Note that



Figure 9. Objects with increasing geometric complexity rendered with subsurface scattering using our algorithm at rates of (a) 26 fps, (b) 26 fps, and (c) 24 fps, respectively. Notice the geometric complexity doesn't greatly influence the render time due to our image-space approach.

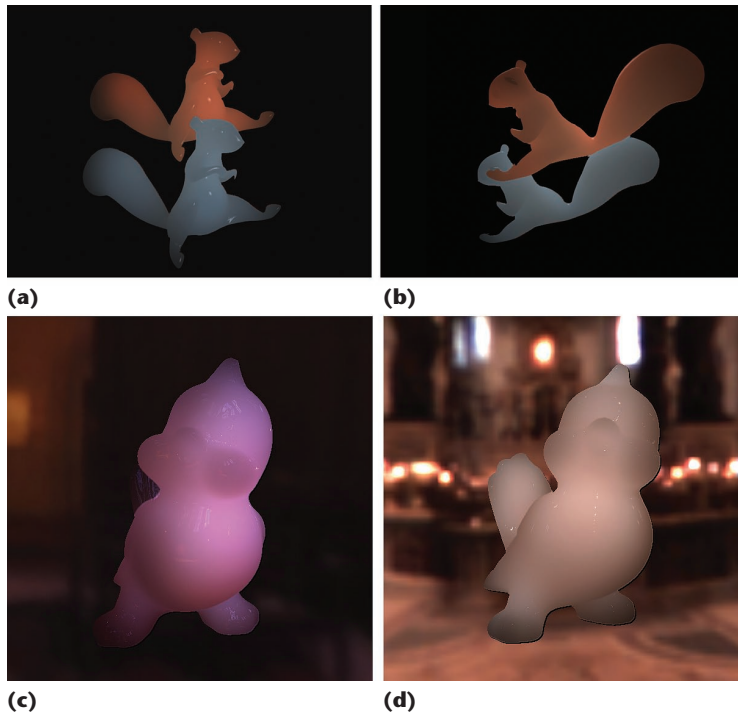


Figure 10. (a-b) Multiple objects with different scattering properties rendered at 25 fps. Note that due to our image-space clipping technique, scattering from one object to the other is prevented even though the objects are adjacent. The bird model rendered using our algorithm under environment lighting: (c) Grace Cathedral (San Francisco) and (d) St. Peter's Basilica (Vatican City).

supporting the multipole approximation doesn't affect our algorithm's real-time performance; it only influences the cost of creating the lookup table. We demonstrate results using the multipole method in our rendering system in Figures 11a–11c.

Analysis

As Figure 1 shows, our algorithm produces results comparable to those obtained using offline renderers. The main difference between our method and most other existing techniques employing 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 exposed to the light source. The quads provide access to points x_o visible to the camera, to which 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 to prevent visual artifacts. In our algorithm, the number of x_o points is determined by the resolution of the rendered image, whereas the number of x_i points is determined by the scattering properties of the material and thresholds that the user chooses. Because the number of points x_i corresponds to the number of splats rendered, it directly influences the rendering speed. Choosing a large threshold will result in too few x_i and pro-

duce patchy artifacts as Figure 12 shows, whereas an excessively conservative threshold will result in a large number of x_i , increasing the render time. We used a threshold of 0.01 (1 percent Weber threshold) in our implementations.

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 corresponds 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. Pixels that fall on the object's surface are converted to sample points x_i , and the blank pixels are discarded. Therefore, the number of points x_i depends on the object's orientation and positioning as observed from the light's view. Thus, for example, when the object rotates, we can see some flickering due to the changing number of x_i . The amount of flickering also depends on the shape of the rendered object. 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. To reduce the appearance of these artifacts, we use a sampling strategy that maintains sampling coherence between frames.

Reducing Temporal Artifacts

To reduce flickering caused by the change in the apparent projected area of the translucent object with respect to the light's view, we must maintain sampling coherence between frames. Recall that for each frame, the object is rendered from the light's view to obtain splat locations. Thus, for every frame, the algorithm obtains a completely new and independent set of splat locations. If the sampling rate isn't high enough, this results in a visible discrepancy in the final rendered images. To enforce coherence between frames, we modify our splat location sampling strategy to reuse certain samples from previous frames. At each frame, the light-view positions textures from both the current and the previous frame are kept. The $n \times n$ splats are rendered twice, first using the positions texture from the current frame and then using the positions texture from the previous frame. However, all samples from each of the two sampling grids aren't splatted. Splat locations are dynamically selected, discarding the unwanted samples by clipping them in the vertex shader. Steps of this sampling strategy are as follows:

Pass 1: Render $n \times n$ splats using light-view positions from the current frame. For each sample, x_i , project x_i into the light's view of the previous frame and

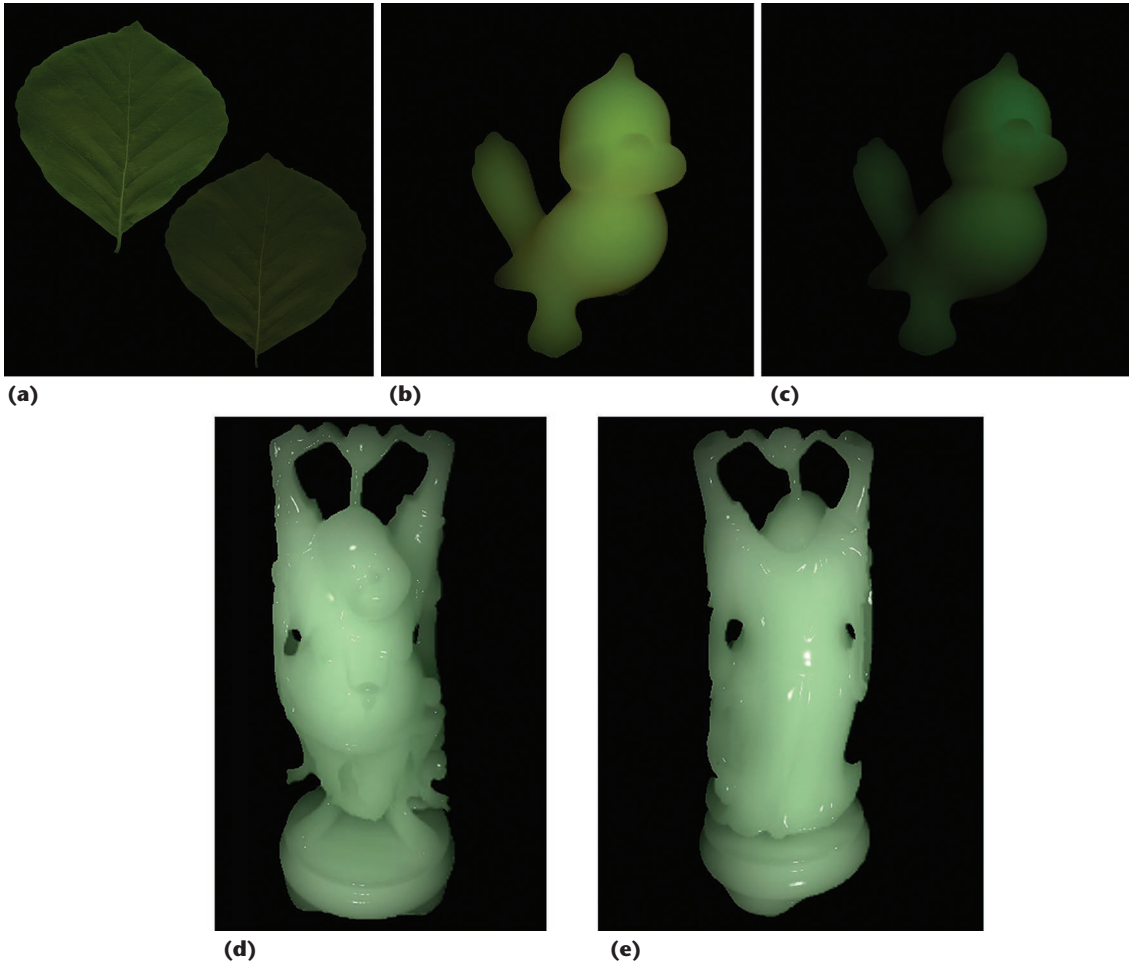


Figure 11. (a–c) Multilayered material rendered using the multipole diffusion method with our algorithm. The material is made of a thin yellow layer and a green layer. (b) and (c) show the effect of increasing and decreasing the thickness of the yellow layer. (d–e) The Happy Buddha model rendered using our algorithm with subsurface scattering properties of jade.

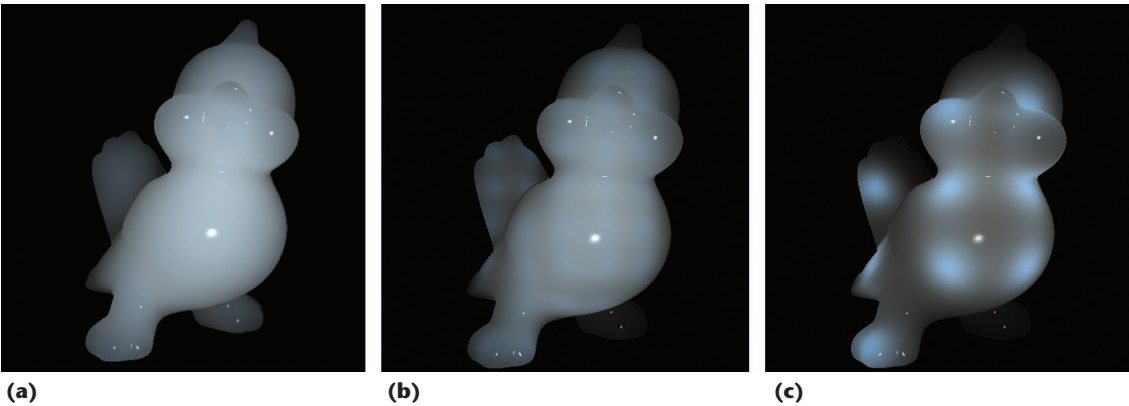


Figure 12. The bird model rendered using (a) 32×32 samples at 31 fps, (b) 16×16 samples at 65 fps, and (c) 8×8 samples at 151 fps. Undersampling results in patchy artifacts because the splats don't overlap sufficiently.

look up the previous positions texture to obtain x'_i . If distance $(x_i, x'_i) < \epsilon$, then clip x_i . Else, splat x_i .


Pass 2: Render $n \times n$ splats using light-view positions from previous frame. For each sample, x'_i , Project x'_i into the light's view of the current frame and look up the current positions texture to obtain x_i . If the depth of x'_i is greater than depth of x_i , then clip x'_i . Else, splat x'_i .

With this sampling technique, all samples from the previous frame that sampled a particular region of the object's surface visible from the light source are retained in the current frame, enforcing sampling coherency between frames. Note that when the object doesn't change its orientation with respect to the light's view, the previous and current frame's position texture is the same. In that case, if the previous sampling technique is

used, all samples in pass 1 will be discarded and all samples in pass 2 will be splatted. When the object moves with respect to the light, all the samples in pass 2 are splatted except the ones no longer visible from the light's view in the current frame.

This new sampling strategy incurs additional vertex-processing cost because two $n \times n$ splatting grids are processed instead of just one. However, the algorithm's bottleneck is the fill rate and not the vertex processing. The total number of samples that actually get splatted remains close to $n \times n$ owing to the clipping of unwanted samples. Therefore, the frame rate isn't significantly affected.

We presented a real-time image-space algorithm for rendering translucent objects using a dipole as well as multipole BSSRDF model. The algorithm computes the area integral using a splatting approach, in contrast to the traditional gathering approach that existing methods use. The splatting approach lets us implement the algorithm entirely on the GPU. The algorithm employs a dual-view representation (light view and

camera view) for storing the incident light and performing the scattering operation, which enables us to account for all the subsurface scattering effects while avoiding artifacts associated with space conversion. Furthermore, this approach is also efficient because it only performs the scattering computation between significant point pairs (emitting and receiving), thus eliminating unnecessary calculations. Our method produces visually convincing results at rates of about 25 fps for arbitrarily complex objects under dynamic view and illumination. In the future, we'd like to accommodate translucent objects with heterogeneous subsurface scattering properties in our real-time rendering system. 

Acknowledgments

This work was supported in part by the I2Lab and I-4 Corridor.

References

1. H.W. Jensen et al., "A Practical Model for Subsurface Light Transport," *Proc. Siggraph*, ACM Press, 2001, pp. 511–518.
2. C. Everitt, *Interactive Order-Independent Transparency*, tech. report, NVIDIA, 2001; http://developer.nvidia.com/object/Interactive_Order_Transparency.html.
3. C. Donner and H.W. Jensen, "Light Diffusion in Multilayered Translucent Materials," *ACM Trans. Graphics (Proc. Siggraph)*, vol. 24, no. 3, ACM Press, 2005, pp. 1032–1039.

Musawir A. Shah is a senior software engineer at NVIDIA. His research interests include real-time realistic rendering of physical phenomena and image-space algorithms. Shah received his PhD in computer science from the University of Central Florida. Contact him at mshah@nvidia.com.

Jaakko Konttinen is a senior software engineer at AMD Graphics. Konttinen received his MS in computer science from the University of Central Florida and is enrolled in the PhD program. Contact him at jaakko@cs.ucf.edu.

Sumanta Pattanaik is an associate professor of computer science at the University of Central Florida. His research interests include realistic image synthesis and real-time realistic rendering. Pattanaik received his PhD in computer science from the Birla Institute of Technology and Science, Pilani. He's a member of the IEEE, ACM Siggraph, and Eurographics. Contact him at sumant@cs.ucf.edu.



Call for Articles

IEEE Pervasive Computing

seeks accessible, useful papers on the latest peer-reviewed developments in pervasive, mobile, and ubiquitous computing. Topics include hardware technology, software infrastructure, real-world sensing and interaction, human-computer interaction, and systems considerations, including deployment, scalability, security, and privacy.

Further details:
pervasive@computer.org
www.computer.org/pervasive

Author guidelines:
www.computer.org/mc/pervasive/author.htm

IEEE pervasive COMPUTING
 MOBILE AND UBIGUITOUS SYSTEMS