APPROVAL SHEET

Title of Thesis: Interactive Rendering of Heterogeneous Translucent Objects

Name of Candidate: Jeremy Michael Shopf
Master of Science, 2007

Thesis and Abstract Approved: _____
Dr. Marc Olano
Assistant Professor
Department of Computer Science
and Electrical Engineering

Date Approved: _____

Jeremy Shopf

University of Maryland, Baltimore County
Department of Computer Science and Electrical Engineering
1000 Hilltop Circle
Baltimore, MD 21250

Voice: (443) 851-1278
Fax: (410) 455-3969
E-mail: jshopf1@cs.umbc.edu
Website: http://userpages.umbc.edu/~jshopf1/

Updated April 4th, 2007

RESEARCH INTERESTS

Computer Graphics and Visualization. Primary interests in interactive computer graphics include global illumination, appearance modeling and volumetric rendering. Application areas of interest are surgical simulation and games.

EDUCATION

Sept 2004 - Dec 2006	M.S. in Computer Science. University of Maryland at Baltimore County. My research is in precomputing light transport due to subsurface scattering in materials of varying composition. *Advisor: Dr. Marc Olano.*
May 2004	B.S. Computer Science. Millersville University. *Advisor: Dr. Roger Webster.*

EMPLOYMENT HISTORY

Jan 2007-Present	**Software Engineer** *3D Application Research Group, AMD, Marlboro, MA.* Member of the Demo team, creating bleeding-edge demo applications showcasing graphics algorithms for marketing and developer relations.
Jun 2006-Aug 2006	**Research Intern** *3D Application Research Group, ATI Research, Inc., Marlboro, MA.* Researching and implementing visual effects for next-generation graphics hardware using DirectX10.
Jan 2006-June 2006	**Research Assistantship** *UMBC VANGOGH Lab, Dr. Marc Olano* Establishing a framework for procedurally defining haptic texture for virtual objects.
May 2005-Aug 2005	**Research Assistantship** National Capital Area Simulation Center, Silver Spring, MD Researching real-time graphics methods for surgical simulation.This includes skin rendering and GPU-accelerated bleeding.
May 2004-Aug 2004 (Full-time) Aug 2004-Jan 2005 (Part-time)	**Internship in Virtual Collaborative Environments** National Capital Area Simulation Center, Silver Spring, MD. TOUCH project
Aug 1999-May 2001	**Web Developer** Expanets, Lancaster, PA. Worked with large development teams to develop many high profile E-commerce sites in ASP.

TEACHING EXPERIENCE

Fall 2006	Teaching Assistant, CMSC435, “Introduction to Computer Graphics,” UMBC
Spring 2006	Teaching Assistant, CMSC411, “Computer Architecture,” UMBC
Fall 2005	Teaching Assistant, CMSC421, “Operating Systems,” UMBC
Spring 2005	Teaching Assistant, CMSC202, “C++ Programming,” UMBC
Fall 2004	Teaching Assistant, CMSC201, ”C Programming,” UMBC

PROJECTS

Rendering Real-time Hetereogeneous Translucent Materials (Masters Thesis)
My thesis entails rendering materials that exhibit translucency and volumetric texture variations (such as skin, marble, etc.). By estimating the physical behavior of light in a participating medium while taking into account the salient perceptual features, perceptually acceptable images are generated. Objects rendered using my method are modeled as a polygonal mesh with a scalar volume describing the spatial density of the material. The interactive performance achieved coupled with the polygonal representation makes this algorithm wellsuited for use in real-time computer graphics applications such as video games.

Acquisition and Relighting of Artistic Shading
Reconstructed the lighting environment of the scene by approximating the geometry of objects in the painting using shape-from-shading methods. Using the approximation to the geometry and the estimation of the lighting, it is possible to alter the lighting environment and also glean the artists interpretation of the objects appearance as a one dimensional function of the angle between the light and the surface normal. Upon obtaining this function, it is possible to reassign this function to objects in other paintings, change the shading of an object in the original painting, or change the lighting environment in a painting by reshading the objects.

Particle Effect Engine Utilizing Geometry Shaders
Developed a particle effect engine for ATI Research's Sushi graphics engine which demonstrates the use of geometry shaders and new stream-out functionality for Microsoft DirectX10. Designed probabilistic algorithms for particle physics behavior. This engine will be utilized in demo applications released as a part of ATI's next generation hardware launch.

GPU-Accelerated Volume Raycaster
Implemented a single-pass algorithm in GLSL using Pixel Shader 3 on NVIDIA 6800 Ultra hardware. The raycaster included a graphical transfer function editor, interactive animation of time-varying data using a playback loop and time slider GUI and the use of non-photorealistic rendering techniques to provide spatial cues.

Efficient Collision Detection and Tissue Deformation in Surgical Simulation
I Developed a stable and efficient method for deforming closed-surface polygonal objects as part of an undergraduate independent study. Surface deformation was calculated based on geodesic distance from actively deformed surface vertex instead of costly/unstable mass-spring models. This research was showcased in Medicine Meets Virtual Reality poster "Using an Approximation to the Euclidean Skeleton for Faster Collision Detection and Tissue Deformations in Surgical Simulators".

TOUCH: Virtual Collaborative Environment for Trauma Training
The main goal of this project was porting the TOUCH project ($\tilde{5}$0,000 lines of code) from Linux to Windows. I implemented an I/O library for the Ascension Flock of Birds magnetic tracker. Porting involved intensive debugging of multi-threaded applications. I wrote a time-trial application to compare user success with various locomotion paradigm. Additionally, I added stereo viewing of the virtual environment for a head mounted display unit.

Advanced Lighting Models
Implemented Banks and Cook-Torrance anisotropic lighting models in GLSL.

Skin Rendering
I implemented real-time simulation of subsurface scattering in skin using GLSL. This was achieve using a two-pass algorithm that precomputes lighting of surface, renders the surface to texture and applies blur to estimate light diffusion in skin.

Visualization of Hurricane Katrina for UMBC Physics Department
Used ray-casting techniques to display relationships of weather data over time. The application interactively animated large weather data volumes. In addition, I implemented wind vector visualization using line integral convolution.

Procedural Haptic Textures
Developed a framework that allows users to define the haptic properties of a surface by writing a C++ procedure. Procedurally definining haptic texture allows dynamic control of the haptic environment through shader parameters.

Realistic bleeding using GPU-based methods
Investigated the application of fluid flow algorithms to the simulation of surgical bleeding. This was part of research assistantship at National Capital Area Simulation Center.

Peer-to-Peer Distributed File System with Digital Rights Management
Written in JAVA using the JXTA P2P API. I chose a structured P2P approach with peers transparently negotiating file system responsibilities.

PUBLICATIONS

Jeremy Shopf, Marc Olano, "Procedural Haptic Texture," UIST 2006. Montreux, Switzerland.

Roger Webster, R. Haluck, R. Shenk, M. Harris, J. Blumenstock, J. Gerber, C. Billman, A. Benson, "Using an Approximation to the Euclidean Skeleton for Faster Collision Detection and Tissue Deformations in Surgical Simulator", Poster Presentation and Proceedings of the Annual Medicine Meets Virtual Reality Conference, (MMVR '2005), Long Beach, California, January 24-29, 2005, pps. 596-598. Name omitted from authors list by error, contact Roger Webster for clarification.

AWARDS

"Best M.S. Research", for "Procedural Haptic Texture". UMBC CSEE Research Review 2006.

3rd place. Greater Baltimore Technology Council MoshPit competition. May 2006. "StreetSmart Traffic" team developed a business plan (including cost and sales projections, business model, etc.) for wifi-enabled GPS devices that collect and utilize traffic information via vehicle-based ad-hoc networking methods.

PRESENTATIONS

Jeremy Shopf, *Procedural Tools for Next and Current Generation Game Platforms* , Game Developers Conference, San Francisco, CA, 2007.

Jeremy Shopf, *Procedural Haptic Texture* , UMBC CSEE Research Review, 2006.

AFFILIATIONS

Association for Computing Machinery (ACM).

UMBC Visualization, Animation, Non-Photorealistic Graphics, Object-Modeling and Graphics Hardware (VANGOGH) Lab.

Abstract

Subsurface scattering is a subtle and often overlooked phenomena responsible for the translucent effects seen in many materials due to the interaction of light with the medium. We describe an algorithm for approximating the appearance of translucent objects with heterogeneous material properties. More specifically, object materials that exhibit translucency and volumetric texture variations. By estimating the physical behavior of light in a participating medium while taking into account the salient perceptual features, we generate perceptually acceptable images. An analysis in terms of performance and rendered images is also performed. Objects rendered using our method are modeled as a polygonal mesh with a scalar volume describing the spatial density of the material. The interactive performance achieved coupled with the polygonal representation makes this algorithm well-suited for use in real-time computer graphics applications such as video games.

Interactive Rendering of Heterogeneous Translucent Objects

by

Jeremy Shopf

Thesis submitted to the Faculty of the Graduate School
of the University of Maryland in partial fulfillment
of the requirements for the degree of
Master of Science
2006

TABLE OF CONTENTS

LIST OF FIGURES

Chapter 1

OVERVIEW

A constant effort in computer graphics is to model complex processes and interactions in the most efficient way possible, resulting in the best ratio of realism to performance. Many of the difficult-to-model processes involve the simulation of light interaction on and within objects in the scene. Complicated reflection and scattering effects produce translucency, shadowing and masking. Properties of the object's surface and interior typically vary across its volume and determine how the light interacts with the material.

Subsurface scattering is the phenomena responsible for translucent materials. When light interacts with a surface, some amount of the light is reflected and the remaining fraction is transmitted beneath the surface. Light is then scattered and absorbed within the object and exits the material at another location. Translucent materials can be divided into two categories: homogeneous and heterogeneous.

Homogeneous material consists of uniform light scattering properties. This category includes milk, wax, plastic, and rubber. Examples of heterogeneous translucent materials are skin, minerals, marble, paper and compositions of homogeneous material.

Previous efforts to model subsurface scattering have resulted in some trade-off between constraints on the physical accuracy of the simulation and performance.

FIG. 1.1. **Top:** Homogeneous translucent material (Milk) **Bottom:** Heterogeneous translucent materials (Mineral)

Several real-time algorithms for computing subsurface scattering have been described [2, 5, 10, 19, 22]. All of these methods are restricted to homogeneous translucent material such as wax. Full participating media simulation methods for rendering non-homogeneous translucent materials (such as photon mapping [13]) are computationally expensive, do not produce anything close to real-time performance, but produce stunningly accurate results. The most computationally inexpensive alternative to full participating media simulation techniques is the shell texture function method described by Chen et al. [3]. This method provides an approximately 1000x improvement in computation time over traditional photon tracing with few distinguishable differences in appearance. However, this method is still not interactive and has a huge memory requirement. A few interactive techniques have attempted to render heterogeneous translucent objects without intensive simulation techniques

and will be discussed in Chapter 3.

By exploiting the highly parallel nature of GPU-based (graphics processing unit) rendering and evaluating only the perceptually significant features of subsurface light transport, we demonstrate that objects with texture variation and subsurface scattering can be rendered at interactive frame rates with very small memory requirements.

Chapter 2

BACKGROUND

2.1 Subsurface Scattering

Traditional models for the scattering of light by a surface are described by the BRDF (bidirectional reflectance distribution function) . Given an incident light direction and a reflectance direction, the BRDF returns the fraction of light that is reflected in the outgoing direction. The BRDF assumes that all light incident on the surface is reflected into the hemisphere above the surface. This is acceptable for metallic surfaces, but the majority of other types of surfaces exhibit some amount of subsurface scattering.

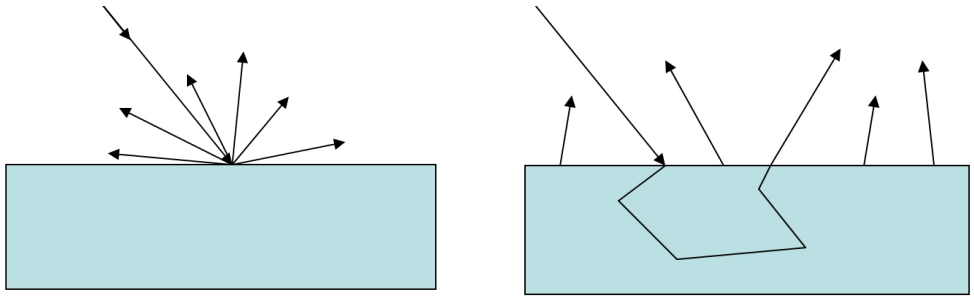

FIG. 2.1. **Left:** BRDF **Right:** BSSRDF (based on [14])

2.2 Participating Media

Evaluating the accurate transport of light through a medium (such as transparent material) requires a participating media simulation. A full participating media simulation requires an evaluation of how discrete representations of light interact with the media. Two techniques for participating media simulations are photon mapping [13] and Monte Carlo ray tracing. Both are extremely computationally prohibitive to real-time applications. Understanding the computation and rendering of participating media requires a review of light transport theory. This is necessary to understand the problem we are trying to solve.

2.2.1 Light Transport Properties

As light moves through a medium, the change in radiance it undergoes is dependent on the absorption (σ_a) and scattering (σ_s) coefficients of the medium. In representing the transport of light with a photon, σ_s and σ_a are the probability that the photon will be scattered or absorbed, respectively. The extinction coefficient (σ_t) is the amount the light is reduced through a medium, which is equal to $\sigma_a + \sigma_s$. Albedo is defined as $\alpha = \frac{\sigma_s}{\sigma_t}$, which is the amount of light diffusely reflected . These coefficients determine the translucent and chromatic properties of a material [26].

2.2.2 Light Scattering in Participating Media

The material properties discussed above are used to describe how light is transported through a medium. Radiance (L) along a light ray through a medium can be changed by several factors at any location x.

First, light can be scattered out of location x in a direction other than the light ray direction. This is referred to as *out-scattering* and is described by the following equation:

$$(\omega \cdot \bigtriangledown)L(x,\vec{\omega}) = \sigma_s(x)L(x,\vec{\omega}) \tag{2.1}$$

Also, the reduction in radiance due to absorption by the medium is:

$$(\omega \cdot \bigtriangledown)L(x,\vec{\omega}) = -\sigma_a(x)L(x,\vec{\omega}) \tag{2.2}$$

The combination of both out-scattering and absorption describes the total loss of radiance at one location:

$$(\omega \cdot \bigtriangledown)L(x,\vec{\omega}) = -\sigma_t(x)L(x,\vec{\omega}), \tag{2.3}$$

where σ_t is the probability of extinction.

In addition to loss of radiance, there will also be an addition of radiance due to scattering from nearby locations in the medium. This is referred to as *in-scattering*. The gain of radiance due to in-scattering is described by:

$$(\omega \cdot \bigtriangledown)L(x,\vec{\omega}) = -\sigma_s(x)\int_{\Omega 4\pi} f(x,\vec{\omega}',\vec{\omega})L_i(x,\vec{\omega})d\vec{\omega}, \tag{2.4}$$

where the incident radiance, L_i, is integrated over the sphere of incoming directions, represented by $\Omega 4\pi$. $f(x,\vec{\omega}',\vec{\omega})$ is the phase function, which describes the amount of light that arrives from direction $\vec{\omega^w}$ and will scatter into direction $\vec{\omega}$. The phase function will be described in more detail in the next section.

Another source of radiance is the emission of radiance from the medium itself (for example, a flame or phosphorous material). This emitted radiance is denoted as L_e. Therefore, the contribution of radiance is described as:

$$(\omega \cdot \bigtriangledown)L(x,\vec{\omega}) = \sigma_a(x)L_e(x,\vec{\omega}). \tag{2.5}$$

By adding equations 2.3, 2.4 and 2.5, the amount of change in radiance per unit length can be calculated as:

$$(\omega \cdot \bigtriangledown) L(x, \vec{\omega}) = \sigma_a(x) L_e(x, \vec{\omega}) - \sigma_t(x) L(x, \vec{\omega}) + \sigma_s(x) \int_{\Omega 4\pi} f(x, \vec{\omega'}, \omega) L_i(x, \vec{\omega}) d\vec{\omega}' \quad (2.6)$$

Jensen integrated Equation 2.6 on both sides for a segment of length s and added the contribution of incoming radiance from the other side of the medium to get the volume rendering equation for participating media [13] :

$$\begin{aligned} L(x, \vec{\omega}) &= \int_{x_0}^{x} \tau(x', x) \sigma_a(x) L_e(x, \vec{\omega}) dx' \\ &+ \int_{x_0}^{x} \tau(x', x) \sigma_s(x) \int_{\Omega 4\pi} f(x, \vec{\omega'}, \omega) L_i(x, \vec{\omega}) d\vec{\omega}' dx' \\ &+ \tau(x_0, x) L(x_0, \vec{\omega}), \end{aligned}$$

where $\tau(x', x)$ is the transmittance along the line segment from x' to x,

$$\tau(x', x) = e^{-\int_{x'}^{x} \sigma_t(\xi) d\xi} \quad (2.7)$$

This equation is very complex, given that it is a five-dimensional function. For rendering applications, this equation is usually abbreviated for performance reasons, as discussed in the Volume Rendering section.

2.2.3 The Phase Function

The phase function defines the distribution of scattering light in the medium. The function is unitless and must integrate to one over the sphere:

$$\int_{\Omega 4\pi} f(x, \omega', \omega) d\omega^{'} = 1$$

Typical phase functions are parameterized by the angle θ between the incoming ray and scattered ray. This angle is typically represented by the average cosine of θ, g, $\in$ [-1,1] , such that -1 is primarily backward scattering, 0 is isotropic (meaning out-scattering from location x is equal for all outgoing directions ω), and 1 is primarily forward scattering. The phase function for an isotropically scattering material is:

$$f(\theta) = \frac{1}{4\pi}$$

2.2.4 Volume Rendering

Volume rendering is a process for constructing a 2D image from a volume containing a mixture of materials. In early volume rendering works, the image was constructed by calculating the absorption of light along the ray through the material to the eye. This simplified model of transport was described by Levoy [20]. This model is known simply as the absorption/emission model. Using a parametric ray equation to describe a viewing ray though the volume:

$$x(s) = x^{'} + s\vec{\omega}$$

the absorption/emission model can be described as:

$$L(x^{'}, \vec{\omega}) = \tau(0, l) L(x, \vec{\omega}) + \int_{x^{'}}^{x} \tau(s, l) E(x(s)) ds, \tag{2.8}$$

where E(x(s)) is the emission term $(\sigma_a(x^{'}) L_e(x^{'}))$ from Equation 2.7. In volume rendering applications, this term is typically calculated by using a simple surface shading model such as Blinn-Phong. The first term in the above equation is used to

calculate the amount of the background color that penetrates the entire volume. The second term calculates the amount of light from each location in the volume along the view ray $\vec{\omega}$ that radiates to point x', accounting for attentuation through all other locations along the ray to the eye.

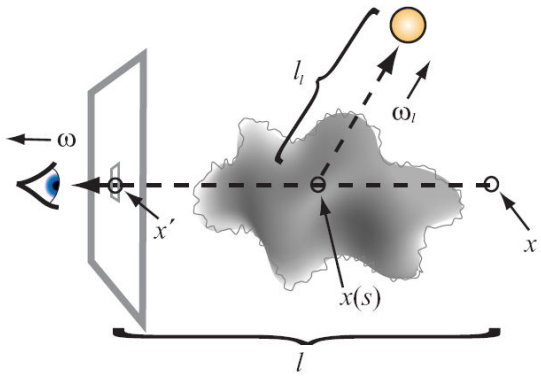

FIG. 2.2. Illustration of vectors involved with ray casting computation (from [16])

A standard volume data structure consists of a 3D structured grid of volume elements (voxels) which contain opacity values in the range of 0.0 to 1.0. These opacity values are similar to the absorption coefficient used in the above equations.

Volumes are rendered using two categories of techniques; Direct and indirect. Indirect methods extract information, such as isosurfaces, to be rendered using traditional computer graphics techniques, while direct methods render from the volume itself. Indirect rendering methods are not suitable for this work so we will not discuss them further.

Drebin et al. and Levoy concurrently provided a model for shading volumes and approximating Equation 2.8 [6, 20]. When a ray enters a voxel the amount of light exiting the voxel x_i in that ray direction is calculated by:

$$C_{out}(v) = C_{in}(v)(1.0 - \alpha(x_i)) + c(x_i) * \alpha(x_i),$$

where $c(x_i)$ and $\alpha(x_i)$ are the color and opacity at the sample location and v is the viewing direction. Therefore, cumulatively applying this equation along equally spaced steps through the volume will give you the pixel color for the final image. Expressed as one equation:

$$C(v) = C(x, y) = \sum_{k=0}^{K} \left[c(x, y, z_k)\alpha(x, y, z_k) \prod_{m=k+1}^{K} (1 - \alpha(x, y, z_m) \right]$$

Therefore, light exiting the volume towards the eye can be calculated by advancing a ray through the volume, accumulating color and opacity values until the the ray has advanced past the boundaries of the volume. Note that in the above equation, only the z value of the ray is increased, because a ray is cast in screen space for each x,y value in the final image. When stated as a procedure:

```
while( (ray has not left the volume) and
       (cumulative opacity < 1.0) )
{
    color += (current voxel color) * (1.0 - cumulative opacity)  *
              current voxel opacity
    cumulative opacity += current voxel opacity *
              (1.0 - cumulative opacity )
    advance ray by step size
}
```

It is important to note that the above volume rendering equation does not account for the scattering of light and also does not account for attenuation of light through the volume when shading volume elements. While this is not physically correct, it simplifies the lighting of the volume and the compositing of the final image. This work serves as the basis for most current direct volume rendering methods.

However, it is simple to add an additional term to the absorption/emission volume rendering equation to account for attenuation of the light when shading voxels. Accounting for this attenuation when shading can also be referred to as shadowing.

$$L(x^{'}, \vec{\omega}) = \tau(0, l)L(x, \vec{\omega}) + \int_{x'}^{x} \tau(s, l)E(x(s))\tau_l(s, l_l)L_l ds, \tag{2.9}$$

where l_l is the distance from the current sample $\mathbf{x}$(s) to the light, and ω_l is the direction to the light. τ_l is essentially the same as τ (Equation 2.7), except that it is accounting for attenuation between the light and the sample instead of the sample and the eye.

Note that this augmented absorption/emission model still does not account for the scattering of light in the volume. This problem has been addressed by Kniss et al. [16] and will be described in the Previous Work section.

2.2.5 Hardware-Accelerated Volume Rendering

More recent methods have worked to accelerate this volume rendering process. Before the advent of the programmable graphics pipeline, it was advantageous to adapt the volume rendering problem to utilize graphics texturing hardware. By arranging a series of view-aligned 2D planes with 3D texture coordinates, the 3D volume can be sampled along the planes and composited into the image plane using simple blend operations. Rotation of the volume is realized by rotating the 3D texture coordinates assigned to the vertices of the planes in the stack (Figure 2.3). This method was initially described by Culipp and Neumann [4] and was further advanced concurrently by several researchers [1, 30]. Many works have been published since then, increasing the performance and image quality possible [7, 15, 18, 21].

Rather than use a slice-based approach, other recent works have taken advantage of the parallel nature of the GPUs (graphics processing units) in modern consumer

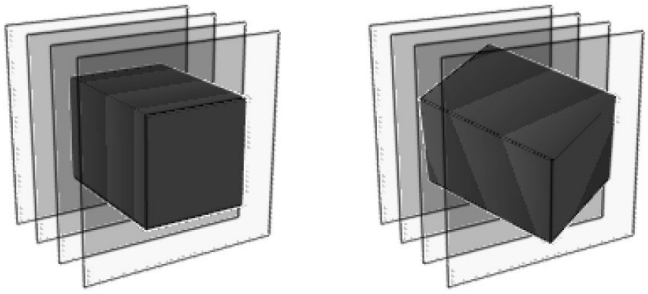

FIG. 2.3. **Left:** Volume in original orientation **Right:** Rotated orientation. View is orthogonal to the slices (from [30]).

grade hardware to perform ray casting in real-time. Ray casting consists of sampling a volume along a direction, starting at some origin. Typically, a ray is cast for each pixel in a generated image in the view direction. Ray casting can be efficient because it allows for more optimizations, such as early ray termination and empty space skipping. Early ray termination ensures that a ray only advances through the volume until an opacity of 1.0 is reached. Empty space skipping techniques permit a ray to "skip over" large regions of space, reducing the number of iterations. These techniques can not be implemented effectively using view-aligned slices.

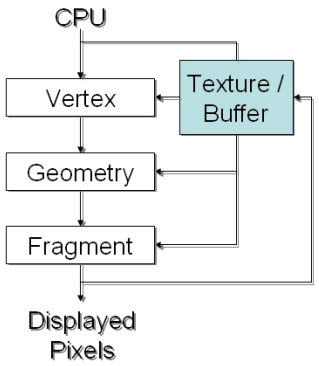

FIG. 2.4. Diagram of GPU pipeline

The *graphics processing unit* (GPU) found in today's commodity graphics hardware provides parallel processing capabilities in the form of vertex, geometry and pixel pipelines (Figure 2.4). Each processing stage has several parallel SIMD processors (exact amount dependent on the hardware), dramatically increasing performance over software processing. Each of these stages are programmable using a relatively short procedure known as a "shader". Vertex shaders are executed for each triangle vertex that is streamed to the pipeline. Vertex shaders typically handle transformations, such as transforming the lighting direction to tangent space, and the assignment of per-vertex attributes, such as color. Geometry shaders operate per-primitive, allowing the programmer to emit or destroy primitives and access adjacency information. The geometry stage of the pipeline has only recently become programmable. Consumer hardware with geometry shader capability is now available. The pixel stage, sometimes referred to as the fragment stage, calculates the resulting color of the generated potential pixels. Pixel shaders permit the implementation of traditional rendering algorithms, such as raytracing, to graphics hardware. There are several advantages to using a GPU-based implementation other than just parallelization:

- Dedicated instructions for graphical tasks
- 4-float vector operations for the cost of a scalar operation
- Fast trilinear texture interpolation
- Fast memory access

We will now describe hardware raycasting, as implemented by Kruger and Westermann [17]. First, the dataset is stored in graphics memory using a 3D texture. 3D textures are indexed with a texture coordinate triplet $(x_{coord}, y_{coord}, z_{coord})$, where $x_{coord}, y_{coord}, z_{coord} \in [0.0, 1.0\rfloor$. The algorithm then proceeds as follows:

- **Pass 1** (Entry-point determination) The front facing faces of the bounding box of the volume, with the 3D texture coordinates encoded as color, are rendered to a texture for access in later stages. The color of each pixel in the resulting texture contains the start position of the ray ($x^{'}$) that will be used for raycasting, in texture coordinate space (Figure 2.5).

- **Pass 2** (Ray direction calculation) The backfacing faces of the bounding box are rendered. Each pixel of the bounding box contains the 3D texture coordinate of where the ray will exit (x, Figure 2.5). By accessing the texture rendered from pass one, the 3D texture coordinate of where the ray entered ($x^{'}$) can be retrieved. Given both x and $x^{'}$, the direction of the ray in texture coordinate space can be calculated: $\omega = \frac{(x-x^{'})}{(x-x^{'})}$. This resulting direction is then rendered to texture as a color for use in the next pass.

- **Pass 3** (Raycasting) Raycasting is now performed iteratively because the information required for the parametric ray equation, $\mathbf{x}(s) = \mathbf{x}_0 + s\omega$. By iteratively increasing s by the stepsize ds, the volume rendering equation can be evaluated.

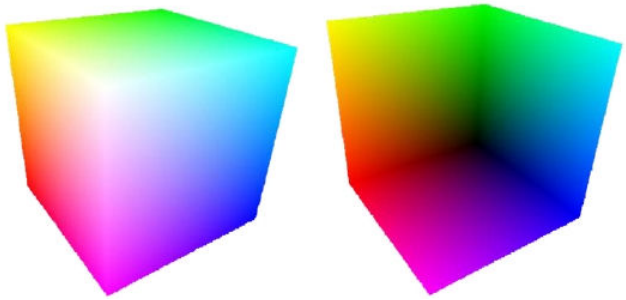

FIG. 2.5. **Left:** Frontfacing texture coordinates **Right:** Backfacing texture coordinates

Chapter 3

PREVIOUS WORK

3.1 Subsurface Scattering

The first general model for subsurface scattering in graphics was presented by Hanrahan and Krueger [1993]. The model accounted for single scattering contributions to surface radiance in layered homogeneous slabs and was designed for use in a ray tracer. Multiple scattering was calculated using Monte Carlo path tracing techniques.

Stam introduced the diffusion approximation for multiple scattering to computer graphics [28]. The diffusion equation was approximated using a multi-grid method for 2D slices. This work is significant in that it provides an analytic method instead of relying on Monte Carlo path tracing techniques.

Jensen et al. provided the first analytical model for subsurface scattering accounting for both single- and multiple-scattering [14]. The dipole diffusion approximation for multiple scattering introduced in this work provides an accelerated method for calculating subsurface scattering elements of surface lighting. The dipole diffusion approximation is the basis for most recent subsurface scattering solutions, so we will explore it further here.

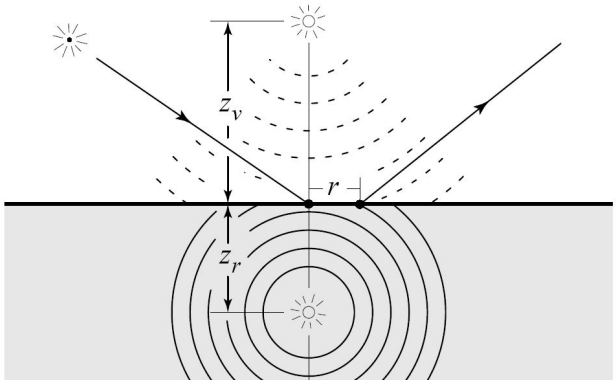

FIG. 3.1. Illustration of the dipole approximation. The dipole approximation places a positive and negative virtual light above and below the surface to estimate subsurface scattering (from [14]).

The dipole diffusion approximation (Figure 3.1 and Table 3.1) relates radiance incident at one surface location to exitance at another surface location. By integrating over the surface, one can estimate the amount of subsurface exitance at surface position x_o. While this solution provides drastic improvements over full participating media simulations (especially in highly scattering material), it still requires expensive integration techniques and is therefore prohibitive to interactive rendering.

Jensen and Buhler later provided a two-pass technique for rendering using the diffusion approximation by separating the calculation of surface irradiance from the calculation of scattering inside the material [12]. Their method offered a 100+ times speedup in rendering time, but still was not fast enough to be used in an interactive application.

3.1.1 Interactive Subsurface Scattering Techniques

Several techniques have been proposed for interactive rendering of homogeneous materials using the diffusion approximation. The majority of these approaches used

$R_d(x_i, x_o) = \frac{\alpha^{'}}{4\pi}\left[z_r(1+\sigma_{tr}d_r)\frac{e^{-\sigma_{tr}d_r}}{d_r^3} + z_v(1+\sigma_{tr}d_v)\frac{e^{-\sigma_{tr}d_v}}{d_v^3}\right]$
$z_r = 1/\sigma_t^{'}$
$z_v = z_r + 4AD$
$d_r = \parallel x_r - x_o \parallel$, where $x_r = x_i - z_r \cdot N_i$
$d_v = \parallel x_v - x_o \parallel$, where $x_v = x_i - z_v \cdot N_i$
A $= \frac{1+F_{dr}}{1-F_{dr}}$
$F_{dr} = -\frac{1.440}{\eta^2} + \frac{0.710}{\eta} + 0.668 + 0.0636\eta$
D $= 1/3\sigma_t^{'}$
$\sigma_{tr} = \sqrt{3\sigma_a\sigma t^{'}}$
$\sigma_t^{'} = \sigma_a + \sigma_s^{'}; \alpha^{'} = \sigma_s^{'}/\sigma_t^{'}$
$\sigma_s^{'}$ = reduced scattering coefficient
σ_a = absorption coefficient
η = relative refraction index
x_i, x_o = in- and out-scattering location
ω_i, ω_o = in- and out- scattering direction
N_i = surface normal at x_i

Table 3.1. Dipole diffusion approximation and related quantities

variations on accelerated radiosity methods.

Lensch et al. provided an interactive method by distinguishing two kinds of subsurface scattering, local and global [19]. Local subsurface scattering, which affects only close by surface points, is estimated by a per-texel filter kernel that is applied to an illuminance texture. Global subsurface scattering, which is defined as light that shines through the object, is represented by precomputed vertex-to-vertex throughput coefficients.

Mertens et al. used a clustered hierarchical radiosity solution to interactively solve the diffusion approximation integral [22]. By calculating form-factors in real-time, they are able to deform translucent polygonal meshes and maintain interactive rates.

While previous methods were CPU-based, Carr et al. adapted subsurface scattering solutions to the GPU [2]. Subsurface scattering is modeled as a single radiosity

gathering step. Rendering is accomplished through a three-pass algorithm that uses links between patches in a multiresolution texture atlas to calculate subsurface scattering.

The method of Hao et al. precomputes the diffusion approximation per-vertex for polygonal meshes under sampled light directions and interpolates between samples at run-time [10]. The original method sampled 200 spherical directions, creating substantial memory requirement. This technique was later updated [11] to use a hybrid representation with sparsely sampled light directions combined with radiance transfer [27].

Translucent shadows maps [5] compute local and global response separately. This technique also computes local response by applying a filter kernel across an illuminated surface and calculates global response by using a technique similar to shadow mapping to determine the distance light has traveled through the medium. By precomputing the dipole approximation for all of the possibilities of incident surface orientations and distances between incident and exitant surfaces, a reasonable approximation of physically accurate subsurface scattering can be achieved. This work inspired our technique presented in this work so it will be further examined in the methodology section.

The translucent shadow map consists of several reference images (Figure 3.2) taken from the viewpoint of the light. The first is a *shadow map*. A shadow map is an image rendered from the viewpoint of a light. Instead of color, the map contains depth values. In normal shadow map uses, the position of each surface in the scene is transformed into light space and its depth is compared with the depth value contained in the shadow map. If the depth is greater than the depth in the shadow map, it must be obscured by another surface; e.g. it is in shadow. However, the depth values in the TSM are used to determine the amount of material the light travels through

between the current point and the light.

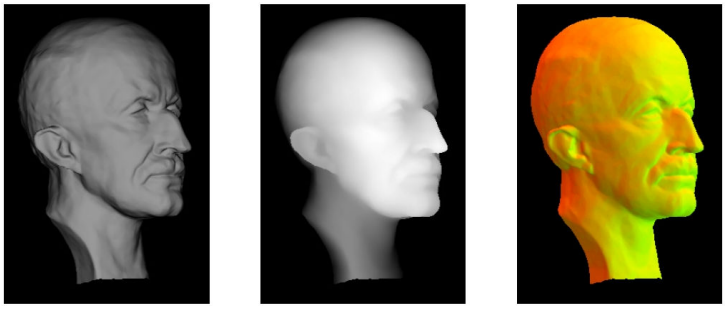

FIG. 3.2. Irradiance, depth and normal components of the TSM (from [5])

Also stored in the TSM are the x and y components of the surface normals visible from the light. The z value can be reconstructed in the shader if need be. Lastly, the irradiance that is transmitted through the surface of the object is stored. The amount of irradiance is equal to the intensity incident on the surface multiplied by the fraction that is transmited according to the Fresnel formula. This fraction can be approximated using Schlick's approximation [25].

The idea behind the TSM method is that it approximates subsurface scattering by sampling the TSM, according to a hierarchical sampling pattern, when rendering points on a surface. To clarify, the subsurface radiance at each point on the surface is calculated by integrating over the surfaces visible in the TSM (Figure 3.3). This integral is approximated by summing the amount of light scattered from each point in the TSM to the point being rendered, using the dipole approximation. As stated earlier, the dipole approximation is not feasibly calculated in real-time.

The TSM method gets around this by precomputing the dipole approximation into a 3D texture for access during rendering. Because the dipole approximation function is 5D, some reduction in dimension must be performed in order to fit it into a 3D texture. This is achieved by quantizing the normal to one dimension. However,

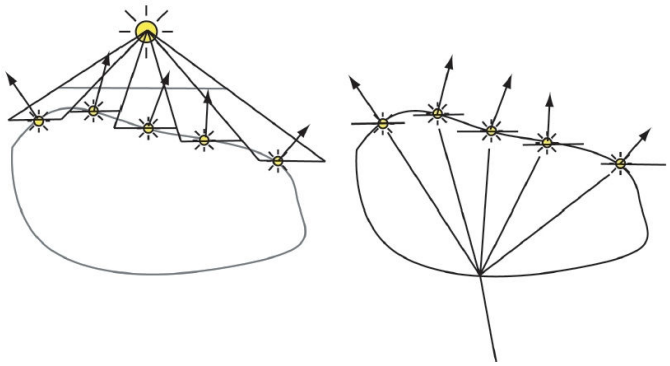

FIG. 3.3. Left: the TSM stores irradiance incident on the surface. Right: Filtered irradiance values are used to calculate radiance exiting at another point

we are not interested in attempting to reconstruct the dipole approximation because it is not necessary to produce realistic images.

Clearly, there has been much work accomplished in rendering interactive homogeneous translucency. Not all translucent materials are homogeneous, however.

3.2 Accelerated Heterogeneous Subsurface Scattering Techniques

Shell texture functions were the first attempt to model the complex light interactions that occur from not only volumetric texture variation in translucent material, but also small surface features (mesostructures) and interreflection between them [Chen et al. 2004].

Chen et al. observed that heterogeneities in the center of an object will have a relatively subtle effect due to scattering in the material [2004]. Therefore, a heterogeneous translucent object can be modeled as a homogeneous core surrounded by a heterogeneous shell. Complicated light interaction was precomputed in a volumetric base volume with photon tracing. This base volume was then synthesized across the surface of the object to create the shell. The irradiance from the core was calculated

using the diffusion approximation. By using these techniques, rendering efficiency was increased by up to 1000x over full participating media simulations. The shell texture function method is not interactive. The objects rendered in their work took an average of ∼89 seconds to render.

3.2.1 Appearance-driven methods

Gosselin et al. presented a simple rendering trick to achieve the look of subsurface scattering effect in skin [24]. The technique first renders a character using standard diffuse lighting. The surface of the character's skin is then "unwrapped" to screen-space and blurred using a kernel defined by a poisson distribution. This blurring simulates the subsurface diffusion of light incident on the skin. This blurred surface texture is then reapplied to the model.

Gosselin's model only accounts for local subsurface scattering. Global subsurface scattering (light that passes through the entire object) is ignored. The radius of the poisson distribution can be altered to account for varying scattering properties but otherwise consideration of physical properties of the material is ignored.

Oat introduced a real-time, appearance-driven technique for multi-layer translucent material rendering that allows for heterogeneous materials [23]. Oat's method represents objects using the core/shell model adopted by the Shell Texture Functions work [3]. The shell of the object is represented by variable-width layers of albedo textures. By utilizing several existing computer graphics techniques, namely normal mapping, transparency masking, parallax offset mapping, and image filtering, images of objects with a semi-transparent volumetric look are created.

This work separately calculates light transmitted into the object and the light transmitted back out of the object. The lighting for the outermost layer of lighting is calculated as simple diffuse lighting with bump mapping. Incoming lighting for each

subsequent layer is a blurred version of the previous layer's lighting, modulated by an opacity map. The blur kernel used is simply a poisson disc with a radius based on the angle of the view vector, the thickness and also the amount of multiple scattering desired in the material. The same iterative blurring function is applied in reverse for back scattered lighting.

It is important to note that the motive of this work was to supply an artist-controllable method for rendering layered shell objects with translucent properties. There is no consideration given to physical properties such as phase, absorption, scattering, or light transmission through the object. The benefit is that the method provides excellent real-time performance ($\sim$60 fps).

Chapter 4

APPROACH

4.1 Discussion of Goals

After reviewing previous work, it is apparent that there exists a void in interactive rendering techniques for displaying objects with heterogeneous translucent appearance features. It is the purpose of this thesis to present a technique that fills this void and to examine its usefulness in terms of performance and visual quality. The ultimate goal is to present a method with a balance of these two criteria.

4.1.1 Salient Translucent Features

Because we wish to generate images that are perceived as translucent but are not necessarily accurate, it is important to understand what makes an object appear translucent. First, objects appear most translucent when lit from behind. It may be intuitive to the reader but it has been confirmed in a user study [9]. It is therefore important that the object look correct when backlit; moreso than when the light is in the same hemisphere of directions as the view direction.

Another feature is the low frequency change in irradiance over the surface of the object. Any high frequencies generated in the rendering process will need to be filtered out. This is due to the diffuse nature of scattering. The further a feature is located through the object material from the viewer, the lower the frequency of the

feature should be.

The relationship between saturation and intensity of color in translucent materials was explored in a user study by Fleming et al. [9]. Objects were rated by users as more translucent when there was a correlation between saturation and intensity, but whether the correlation was positive or negative was of no consequence. It is therefore our desire to maintain this relationship.

4.2 Overview of Method

In this thesis, we present a method for interactively rendering heterogeneous translucent materials. This method represents an object in a rendered scene with a polygonal surface representation and a scalar volume representing the spatial density of material in the object (Figure 4.1). The extent of the volume is the bounding box of the polygonal representation. We have chosen this hybrid representation for several reasons. First, polygonal mesh representation is standard for real-time applications such as games. Having a discrete boundary representation is essential for collision detection, allows a high resolution surface for shading and bumpmapping, reduces memory requirements by using a low resolution volume for spatial density inside of the surface, and can also accelerate volume rendering techniques.

4.3 Heterogeneous Translucent Shadow Maps

We have designed a two-pass algorithm for the approximation of light transport in a heterogeneous translucent object. In the first pass, we compute illumination incident on all surfaces on the object. In the second pass, we approximate how this illumination scatters through the object to visible surfaces. To maintain information about surfaces that are receiving light, we store several reference images from the viewpoint of the light source. To compute subsurface scattering, we sample these

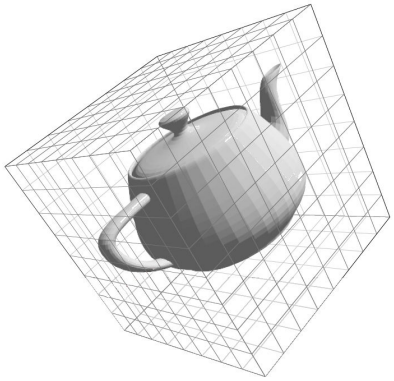

FIG. 4.1. Hybrid polygonal/voxel representation

reference images in a hierarchical sampling pattern. This reference image sampling technique is similar to that presented to the translucent shadow map method [5].

In our method, we do not store depth information in the reference images. For our ray casting method that we use in conjuction with the TSM method, we require 3D texture coordinates to acquire ray directions in texture coordinate space. We are able to acquire the same depth information available in a shadow map from the texture coordinates. The reasoning for the use of texture coordinates instead of scalar depth values will become more clear once the ray casting method is discussed.

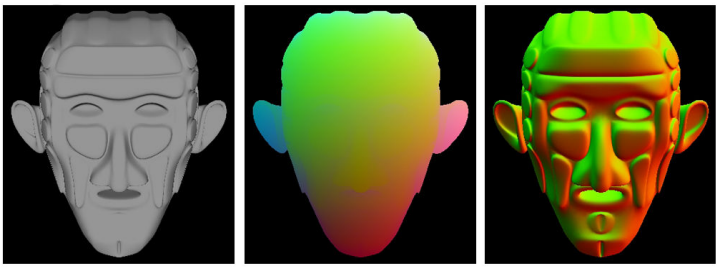

FIG. 4.2. Irradiance, texture coordinate, and normal components of HTSM. Compare with Figure 3.2.

Our method supports both distant and point light sources. When using distant light sources, the reference images can be rendered from a minimum distance along the light direction that captures all lit surfaces. In this manner, it is ensured that the object maintains the maximum amount of resolution in the HTSM. For point sources, the reference images must be rendered from the position of the light. Therefore the object will achieve less accurate results as the light moves away from the object. This problem is common in shadow map algorithms and there are methods to improve upon the naive approach [8, 29], but we do not implement them. We mask this issue by attenuating the intensity of the point light illumination so that no light is reaching the object when the source reaches a problematic distance.

4.3.1 Rendering

Rendering using the HTSM method consists of evaluating the sampling pattern in Figure 4.3 for each point that is rendered on the surface. For each sample, the orientation of the surface and distance between the sample and the point on the surface being evaluated is used to estimate the amount of radiance that the sample contributes to the point being rendered.

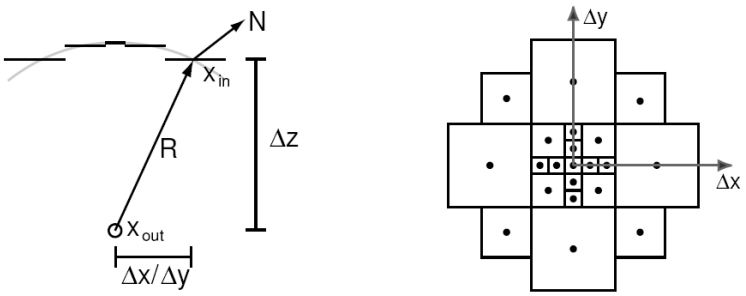

FIG. 4.3. **Left:** Side view of HTSM sampling pattern **Right:** HTSM sampling pattern

The hierarchical sampling method is used because it is infeasible to access the 3D texture for every pixel in the HTSM map. In turn, the HTSM method uses samples representing increasing areas as the radius increases. This is achieved by using lower mipmap levels of the HTSM. A mipmap is a stack of increasingly lower resolution versions of the original image. Each level has half of the resolution of the level before it. Therefore each texel in a mipmap level is the average of four texels in the level above it. By using lower levels in the mipmap chain, the HTSM method can approximate larger areas of the object surface.

4.3.2 Evaluating Material Density

Simply evaluating the samples from the HTSM assumes that the material between the samples and the point being rendered is homogeneous. Instead, our algorithm defines a base material and the scalar density volume defines the amount of material in each voxel. Any radiance entering a voxel with the absorption coefficient σ_a and density value $\alpha_d \in [0, 1]$ will be subject an adjusted absorption coefficient $\sigma_{at} = \sigma_a + \alpha_d$. In order to account for this additional extinction of radiance, the extinction integration function (Equation 4.1) must be adjusted:

$$\tau(x^{'}, x) = e^{-\int_{x'}^{x} \sigma_{at}(\xi) d\xi} \tag{4.1}$$

The ray casting method we use to integrate the extinction between the point being rendered and each sample in the hierarchy is a GPU-based method based on a previous algorithm [17]. To summarize, the algorithm renders the 3D texture coordinates of the front and back faces of the bounding box and subtracts corresponding pixels in both images to get a final image that is the direction texture. This texture contains the direction of each view ray in texture coordinate space.

We alter this ray casting algorithm. Because we are interested in how far light

has traveled from where it entered the object to where it exits, we first render the texture coordinates of the polygonal object from the view of the light (this is stored in the HTSM, as described earlier). Then, the texture coordinates of the front faces of the object from the viewpoint of the camera are rendered. To find the sample ray directions for a pixel, the pixel's position is transformed into light space and the 3D texture coordinates of all of the samples are retrieved from the HTSM. The 3D texture coordinate of the current point is subtracted from each of the sample 3D texture coordinates to get the ray directions used for raycasting. Figure 4.5 illustrates this operation for the first sample ray.

We have chosen a fixed number of iterative steps for the sampling rays. Because of the SIMD nature of GPUs, it is inefficient to adapt the number of steps per ray. Recent GPUs process pixels in blocks of between 16 and 1024, depending on the card and manufacturer. As a result, if one sample ray in a block takes fifteen steps and the rest all take ten, all of the rays in the block have to perform the amount of work for fifteen steps.

Because our algorithm uses a small number of steps, aliasing can be issue. Care must be taken when selecting a density volume size and the frequency of features contained within. This would be less of an issue if graphics hardware could support mipmap filtering for 3D textures.

4.3.3 Approximating Chromatic Absorption

Traditional volume rendering techniques rely on two parameters, attenuation and material color. Participating media simulation requires different parameters: absorption, scattering and phase. While attenuation is similar to absorption in that they account for the reduction in irradiance through material, they are proportional not equal. Attenuation has a value between 0.0 and 1.0 but absorption can have a

value between 0.0 and infinity. Note that it is impossible for a material to exhibit no absorption properties, therefore it is impossible for a material to have an absorption coefficent of exactly zero. Attenuation is usually specified by an *alpha* value:

$$\alpha = 1 - e^{-\tau(x)} \tag{4.2}$$

Attenuation is typically specified as an achromatic value, meaning that all wavelengths of the incoming radiance are absorbed equally. This is contrary to how radiance is handled in participating media simulations. Absorption and scattering coefficients are specified separately for red, green and blue wavelengths. It is possible to specify a separate alpha for each wavelength i:

$$\alpha_i = 1 - e^{-\tau_i(x)} \tag{4.3}$$

where $i \in r, g, b$.

Consider a material that has an $\alpha = (0.02, 0.02, 1.0)$. As light bounces through the material, the blue wavelength will become completely reduced, leaving mostly red and green wavelengths (yellow). This yellow color can be thought of as the *transport color* [16]. It is the color produced as a result of chromatic absorption. This transport color is equal to $(1.0, 1.0, 1.0) - \alpha_i$.

In our algorithm, the user specifies the transport color C_t and the distance through the base material light travels before it exhibits this color. In this manner, we can estimate the radiance transmitted through the object between two points on the surface, accounting for absorption.

4.3.4 Approximating Phase

The phase function $f(x, \omega', \omega)$, when given an incoming light direction ω' at location x, describes the fraction that out-scatters in direction ω. This can be taken into account when integrating radiance through the object. Each sample ray can be weighted by an appropriate fraction considering its direction.

Phase functions can be parameterized by the angle between ω and ω'. In our algorithm, we consider ω' as the sample ray direction and ω as the negated normal of the surface at x on the surface.

Rather than evaluate the phase function during execution, we precompute the phase function as a 1D texture. During run-time, we compute the angle between ω and ω' using the dot product to index this texture. Because the range of values of the dot product between these two directions is [-1,1] and the texture coordinates for a 1D texture are [0,1⌋, we scale the dot product result by .5 and bias by .5. This value can then be used for each ray to calculate the appropriate weighting for the final irradiance at location x.

Our translucent rendering algorithm calculates the transport through the object in a strictly forward fashion. Unfortunately this makes it impossible for the algorithm to model objects that exhibit some visual effects that result from back scattering.

4.3.5 Putting It All Together

Table 4.3.5 contains the final equation and all related variables for calculating radiance at the surface in pixel position x,y. Note that the $diffuse$ and *specular* terms should be zero when the dot product of the light and $N_{x,y}$ and when the surfaces are occluded.

To summarize our algorithm:

```
-Render the HTSM reference images from the light
```

$L(x,y) = T_c * \frac{1}{n} \sum_i^n (\frac{2T_d - (D_i + \alpha_i)}{T_d} * f(\theta_i)) + Diffuse + Specular$
T_c = Transport color
T_d = Transport distance
D_i = Length of sample ray i
α = Accumulated absorption along ray i θ_i = Angle between sample ray i and $-N_{x,y}$
$N_{x,y}$ = Normal at x,y
$f(\theta_i)$ = Phase function evaluated for θ_i
n = Number of sample rays

Table 4.1. Our light transport approximation and related variables.

```
For each visible surface:
    -Find the surface position in light space
    For each sample in the hierarchical sampling pattern:
       -Fetch the lit surface from the HTSM with the same
          x,y position in light space
       -Fetch the irradiance for the lit surface
       -Store the distance between these two surfaces
       -Construct a ray from the visible surface to this surface
       -Ray cast using the constructed ray to accumulate density
       -Based on the distance between the surface, the amount of
          density, and the phase, add the amount of irradiance
          from this ray direction
   Set the color of the visible surface equal to the radiance from all rays
   Add surface shading (if the surface is lit)
```

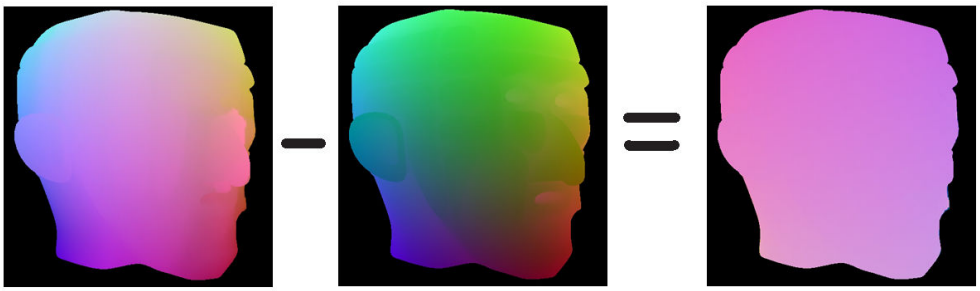

FIG. 4.4. From Left: Projected 3D texture coordinates from the HTSM (END), 3D texture coordinates of front faces (START), DIR = END-START. Coordinates are scaled and based to show all values.

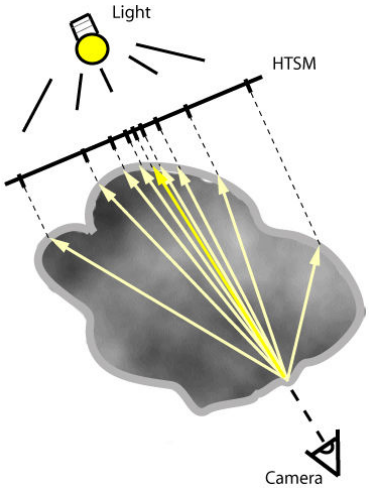

FIG. 4.5. The position on the surface being viewed is transformed into screenspace and all rays are calculated from the 3D texture coordinates in the HTSM sampling hierarchy. Ray casting is then performed to accumulate density in the interior of the object.

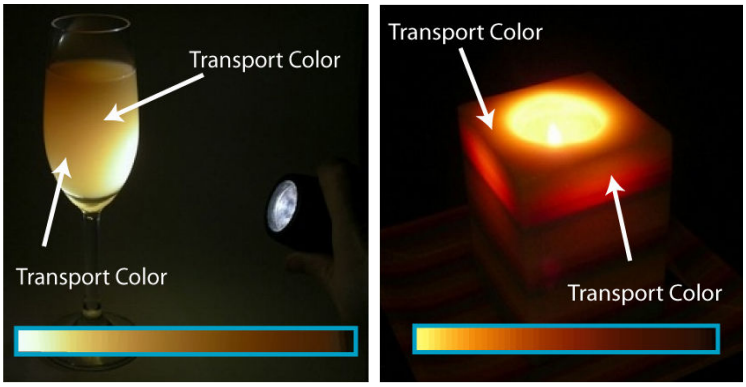

FIG. 4.6. Photographs depicting transport color in juice and a candle.

Chapter 5

RESULTS

Our method provides a real-time approximation to the complex subsurface scattering problem. In this chapter, we analyze images created using our approximation and their strengths and weaknesses. We also discuss the performance of our algorithm on consumer-level graphics hardware.

5.1 Discussion

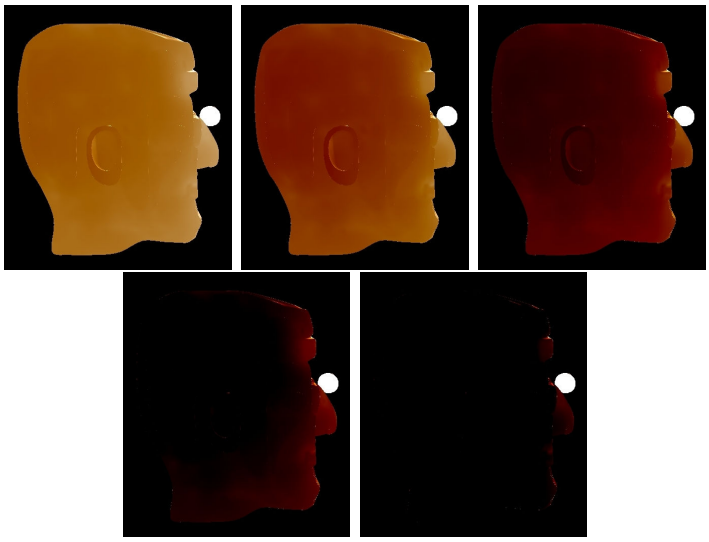

FIG. 5.1. Mayan head rendered with varying absorption properties

As depicted in Figure 5.1, varying the absorption properties of the material drastically changes the appearance of the object. When absorption is increased, the heterogeneous interior of the object becomes much more pronounced. This effect is expected in real materials. When absorption is low, the effects of multiple scattering become much more pronounced. The diffuse nature of multiple scattering blurs out detail of the interior.

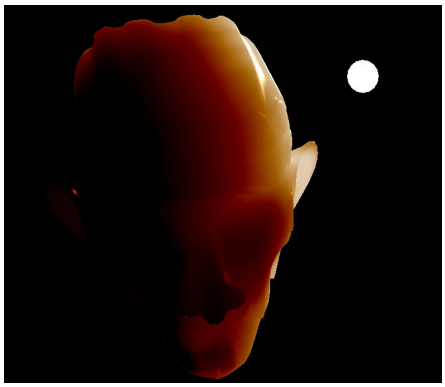

FIG. 5.2. Oblique angle of candle wax mayan head

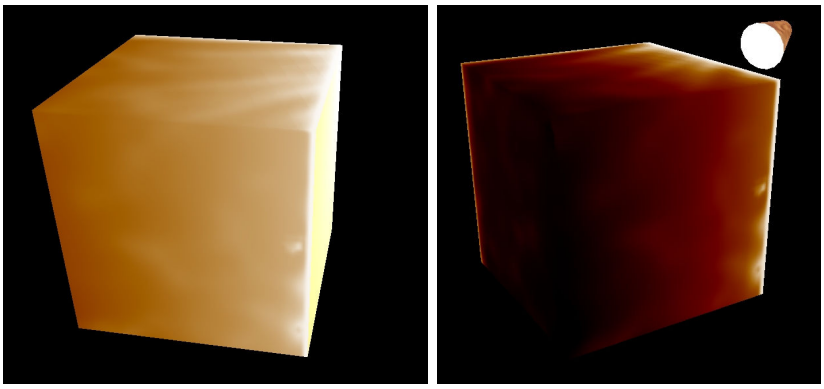

FIG. 5.3. Cube rendered with different absorption properties. Note rim lighting from scattering along surface.

Figures 5.2 and 5.3 further explore the use of the wax transport color for different meshes and varying absorption properties.

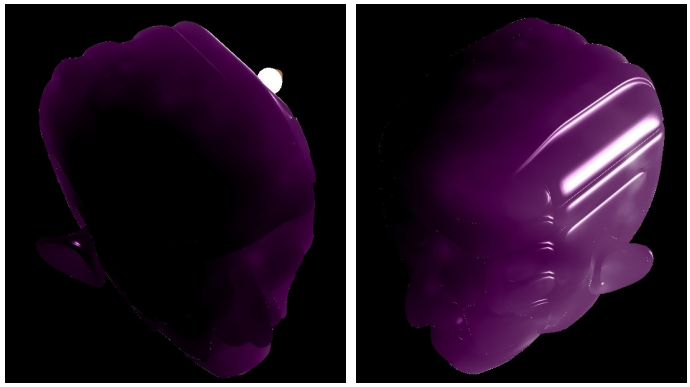

FIG. 5.4. Translucent plastic model under different lighting conditions.

Figure 5.4 depicts a model with a translucent purple plastic material. Note that the achieved effect here is that the scattering in the material is mostly single scattering because it appears "harder" and less diffuse. This is because we have not saturated the high intensity colors on the lit surface. This reinforces the relationship of translucency and the correlation between intensity and saturation discussed earlier.

FIG. 5.5. Car rendering using varying lighting directions. Note the smooth transition between directly lit and obscured surfaces.

In Figure 5.5, we demonstrate another purple translucent material. It is important to notice that there are no noticable artifacts between directly lit surfaces

and surfaces that are illuminated solely by subsurface scattering. Other subsurface scattering algorithms such as the Translucent Shadow Maps method [5] separate the calculation of local and global subsurface scattering and then combine them in the final image. This can cause surfaces with orientations nearly orthogonal to the light direction to appear darker than surfaces that are completely obscured, which is visually distracting.

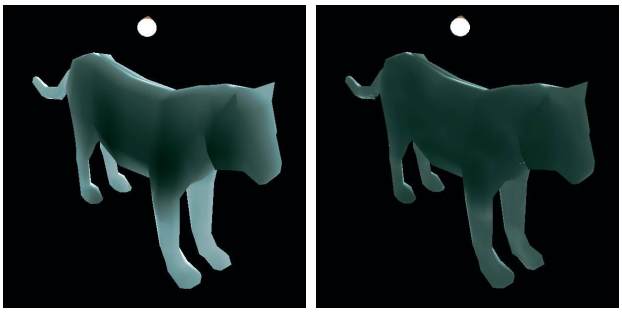

FIG. 5.6. Comparison of surface radiance between homogeneous and heterogeneous versions of the low-poly tiger model

Figure 5.6 we illustrate the stark difference in the surface radiance when accounting for heterogeneous material. When ignoring the volumetric texture variation, the radiance is merely a function of the width of the object that the scattered light travels through.

In Figure 5.7, we provide the visual result of increasing the number of samples used for each pixel. The zoomed images focus on a particular high-frequency feature that is present on the nose of the model. This feature is not desired because it would not be present on an object with significant multiple scattering. The feature would only be this stark if the only light reaching this point through the object had come directly through the entire object without scattering. By using more rays, absorption by the material in a larger cone of directions can be accounted for. It

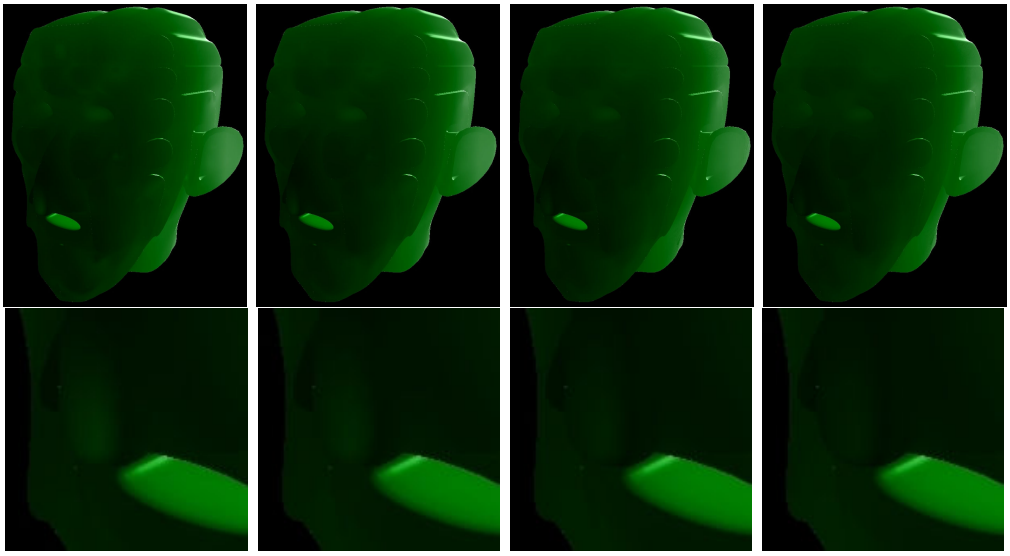

FIG. 5.7. Left to Right: 1, 5, 9, 13 sample rays from the HTSM. Bottom: Close-up of nose region. Note the gradual smoothing of the light region.

may be possible to get more of a blurring effect with fewer rays by using a higher mip level for density volume accesses that are for locations that are further from the viewer. Unfortunately, none of the graphics hardware available to the author support mipmapped three-dimensional textures.

Figure 5.8 shows a translucent object with a more familiar internal structure. The banding artifacts are a result of too few sample rays. The shape would appear much smoother with an increased number of sample rays.

5.2 Performance

We have implemented this work on an ATI Radeon X1600, a previous generation graphics card. The algorithm is essentially performed in two-passes: Render the 3D texture coordinate and irradiance reference images of the object from the light view and then use these reference images to ray cast the density volume to produce the

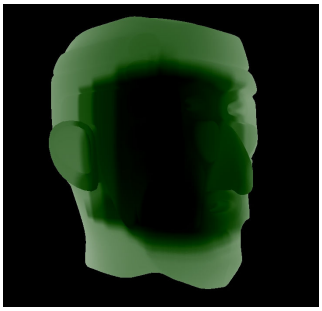

FIG. 5.8. Translucent object with an embedded cube shape, rendered with 5 sample rays.

final image.

Considering that each pixel on the surface of the object must ray cast n rays, where n is the number of samples in the HTSM sampling hierarchy, and each ray must access the 3D density texture for each step it is advanced, it becomes apparent that this algorithm is texture-access bound. This is a common problem in graphics hardware. A texture access in a shader delays all computation based on the result of that texture data fetch. The current magic number in graphics hardware is about eight math instructions can be per texture access. The problem is compounded when one texture-fetch is dependent on another. Consider a sample hierachy of nine samples. Then consider that each sample ray takes five steps through the density volume. That is forty-five texture accesses for each pixel on the surface.

Graphics card	n=1	n=5	n=9	n=13
NVIDIA 7900GTX SLI	300fps	105fps	41fps	26fps
ATI Radeon X1900XT	132fps	55fps	30fps	15fps
ATI Radeon X1600	120fps	42fps	23fps	12fps

FIG. 5.9. Rendering frame rates for different hardware.

Memory requirements for our method are very modest. To store all reference images and also a 3D texture (64x64x64) containing density information requires only ∼3MB. Compared with the Shell Texture Functions method (≥159MB) the requirement is quite reasonable [3].

5.3 Comparison

Method	Image/Geo. Based	Homo	Hetero	Precomp. / Runtime	Deform.
Lensch	G	Yes	No	Runtime	No
Mertens	G	Yes	No	Runtime	Yes
Carr	G	Yes	No	Runtime	No
TSM	I	Yes	No	Runtime	No
Hao	G	Yes	Yes (in shell)	Pre	No
Sloan	G	Yes	Yes	Pre	Partial
HTSM	I	Yes	Yes	Runtime	Partial

FIG. 5.10. Comparison table for existing subsurface scattering algorithms.

It is difficult to perform a quantitative comparison between existing methods and our method. Older methods were performed on obselete hardware and are also sufficient difficult to implement. We therefore provide a qualitative comparison.

First, our method is image-based. This is a benefit because the complexity of the algorithm is not affected by the polygon count in the meshes used. We also support both homogeneous and heterogeneous translucent material. In addition, we evaluate all light transport at run-time, which distinguishes us from the other two heterogeneous subsurface scattering methods. This is a benefit because it permits the run-time adjustment of material properties, eliminates the need for lengthy precomputation time and allows certain deformations.

When examined qualitatively, it is apparent that our technique has significant advantages over all existing methods.

5.4 Limitations

FIG. 5.11. It appears that the back of the head is creating shadows on the ears, but what is actually happening is that the light is being attenuated in the space between the head and ears, as if there were material in that space.

There are some assumptions that have been made regarding the geometry of the objects rendered and also the behavior of light. Because all light transport is modeled as transfer between surfaces visible from the light and surfaces visible from the eye, it is assumed that all space between these surfaces are inside the object. Therefore the algorithm assumes a convex object. Some artifacts may arise (as in Figure 5.12) but typically they are not very distracting or displeasing in objects that are mostly convex.

Also, because all reference images are rendered from the light's viewpoint, the algorithm only supports point and distant lights, not environment lighting. As mentioned earlier, this algorithm only computes scattering as forward-scattering process.

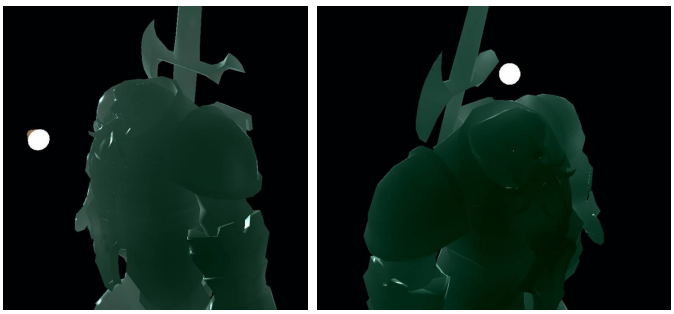

FIG. 5.12. A complicated model that violates the convex object limitation of our algorithm. Observe that the lit object suffers no perceptually unsettling artifacts.

Chapter 6

CONCLUSION

Our algorithm is essentially a two-pass method. We first obtain information about the surfaces which are receiving illumination. This includes the amount of irradiance transmitted and 3D texture coordinates. As a second pass, we query this information to estimate how the incident illumination on the lit surfaces is scattering to surfaces that are visible from the user's viewpoint.

To produce the perceptually-correct effects from the subsurface scattering, we allow the user to define an intuitive transport color which defines the color the scattered light becomes as it is absorbed. The user also defines a distance through the material that the light travels before exhibiting the transport color, assuming the material is homogeneous. This distance in combination with the transport color describes the absorbent properties of the material.

To account for the density changes in the interior of the object, we perform a coarse ray casting of a scalar volume texture. Rather than simply ray casting *through* the object as is done in standard volume rendering techniques, several rays are cast from points on visible surfaces to lit surfaces. This algorithm is implemented on the GPU to accelerate the rendering process.

In summary, we have presented a method for rendering heterogeneous translucent objects in real-time. This is the first method to account for heterogeneities

in the interior of an object when calculating surface irradiance due to subsurface scattering. This is important because until now an entire class of materials have been excluded from the real-time rendering arena. By considering the physical implications of light transport and also the characteristics we expect from translucent materials, we produce convincing images at very reasonable real-time rates. Considering the performant frame rates achieved, low memory requirement and hybrid polygonal/volumetric representation, our work represents a significant step forward in real-time rendering of translucent objects.

Bibliography

[1] Cabral, B., Cam, N., and Foran, J. Accelerated volume rendering and tomographic reconstruction using texture mapping hardware. In *VVS '94: Proceedings of the 1994 symposium on Volume visualization* (1994), pp. 91–98.

[2] Carr, N. A., Hall, J. D., and Hart, J. C. GPU algorithms for radiosity and subsurface scattering. In *HWWS '03: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware* (2003), Eurographics Association, pp. 51–59.

[3] Chen, Y., Tong, X., Wang, J., Lin, S., Guo, B., and Shum, H. Shell texture functions. In *ACM Transactions on Graphics* (2003), vol. 23, pp. 343–353.

[4] Culipp, T., and Neumann, U. Accelerating volume reconstruction with 3D texture hardware. Tech Report TR93-027, University of North Carolina, Chapel Hill, 1993.

[5] Dachsbacher, C., and Stamminger, M. Translucent shadow maps. In *EGRW '03: Proceedings of the 14th Eurographics workshop on Rendering* (2003), Eurographics Association, pp. 197–201.

[6] Drebin, R. A., Carpenter, L., and Hanrahan, P. Volume rendering. In *SIGGRAPH '88: Proceedings of the 15th annual conference on Computer graphics and interactive techniques* (1988), pp. 65–74.

[7] Engel, K., Kraus, M., and Ertl, T. High-quality pre-integrated volume rendering using hardware-accelerated pixel shading. In *HWWS '01: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware* (2001), pp. 9–16.

[8] Fernando, R., Fernandez, S., Bala, K., and Greenberg, D. P. Adaptive shadow maps. In *SIGGRAPH 2001, Computer Graphics Proceedings* (2001), ACM Press, pp. 387–390.

[9] Fleming, R. W., Jensen, H. W., and Bulthoff, H. H. Perceiving translucent materials. In *APGV '04: Proceedings of the 1st Symposium on Applied perception in graphics and visualization* (New York, NY, USA, 2004), ACM Press, pp. 127–134.

[10] Hao, X., Baby, T., and Varshney, A. Interactive subsurface scattering for translucent meshes. ACM Symposium on Interactive 3D Graphics, 2003.

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

[12] Jensen, H. W., and Buhler, J. A rapid hierarchical rendering technique for translucent materials. *ACM Transactions on Graphics 21* (2002), 576–581.

[13] Jensen, H. W., and Christensen, P. Efficient simulation of light transport in scenes with participating media using photonmaps. In *Proceedings of ACM SIGGRAPH 1998* (1998), pp. 311–320.

[14] Jensen, H. W., Marschner, S. R., Levoy, M., and Hanrahan, P. A practical model for subsurface light transport. In *Proceedings of ACM SIGGRAPH 2001* (New York, 2001), Computer Graphics Proceedings, ACM Press/Addison-Wesley Publishing Co., pp. 511–518.

[15] Kniss, J., Kindlmann, G., and Hansen, C. Multidimensional transfer functions for interactive volume rendering. *IEEE Transactions on Visualization and Computer Graphics 8*, 3 (2002), 270–285.

[16] Kniss, J., Premoze, S., Hansen, C., Shirley, P., and McPherson, A. A model for volume lighting and modeling. *IEEE Transactions on Visualization and Computer Graphics 9*, 2 (2003), 150–162.

[17] Kruger, J., and Westermann, R. Acceleration techniques for GPU-based volume rendering. In *Proceedings of the 14th IEEE Visualization 2003* (2003), vol. 38.

[18] LaMar, E., Hamann, B., and Joy, K. I. Multiresolution techniques for interactive texture-based volume visualization. In *VIS '99: Proceedings of the conference on Visualization '99* (1999), pp. 355–361.

[19] Lensch, H., Goselle, M., Bekaert, P., Kautz, J., Magnor, M., Lang, J., and Seidel, H.-P. Interactive rendering of translucent objects. In *Proceedings of IEEE Pracific Graphics* (2002), pp. 214–224.

[20] Levoy, M. Display of surfaces from volume data. *IEEE Comput. Graph. Appl. 8*, 3 (1988), 29–37.

[21] Meissner, M., Hoffmann, U., and Strasser, W. Enabling classification and shading for 3D texture mapping based volume rendering using OpenGL

and extensions. In *VIS '99: Proceedings of the conference on Visualization '99* (1999), pp. 207–214.

[22] Mertens, T., Kautz, J., Bekart, P., and Reeth, F. V. Interactive rendering of translucent deformable objects. In *Proceedings of the 14th Eurographics Workshop on Rendering* (2003), pp. 130–140.

[23] Oat, C. Rendering gooey materials with multiple layers. In *Proceedings of SIGGRAPH 2006 course notes, course 26, Advanced Real-Time Rendering in 3D graphics and Games* (2006), pp. 71–79.

[24] Sander, P. V., Gosselin, D., and Mitchell, J. L. Real-time skin rendering on graphics hardware. In *Proceedings of ACM SIGGRAPH 2004 Sketches* (2004).

[25] Schlick, C. A customizable reflectance model for everyday rendering. *Eurographics Workshop on Rendering* (1993), 73–84.

[26] Siegel, R., and Howell, J. R. *Thermal Radiation Heat Transfer, 3rd Edition*. Hemisphere Publishing Corporation, New York, 1992.

[27] Sloan, P.-P., Kautz, J., and Snyder, J. Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 2002), ACM Press, pp. 527–536.

[28] Stam, J. Multiple scattering as a diffusion process. In *Rendering Techniques '95 (Proceedings of the Sixth Eurographics Workshop on Rendering)* (New York, NY, 1995), P. M. Hanrahan and W. Purgathofer, Eds., Springer-Verlag, pp. 41–50.

[29] Stamminger, M., and Drettakis, G. Perspective shadow maps. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques* (2002), pp. 557–562.

[30] Wilson, O., Gelder, A. V., and Wilhelms, J. Direct volume rendering via 3D textures. Tech Report UCSC-CRL-94-19, University of California at Santa Cruz, 1994.