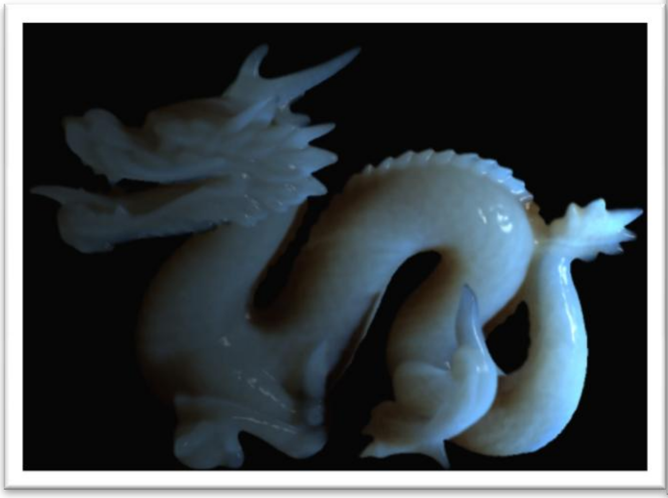
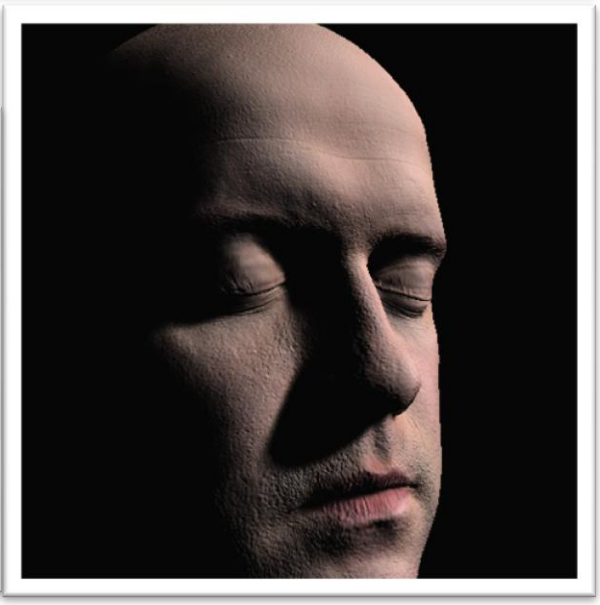
IMM - DTU

Real-time Scattering





Abstract

For a long time, real-time computer graphics have been limited to simulating local illumination with the only completely dynamic global illumination effect being shadows. Recent work is producing results allowing more and more global illumination effects to become plausible. Especially diffuse interreflection and light passing through scattering media is receiving a lot of attention.

This thesis aims to analyze and implement solutions for various aspects of light scattering calculations. Both participating media, and subsurface scattering has been evaluated, and different solutions have been implemented.

Erik Rune Skals Livermore

s042572

28. February 2012

TABLE OF CONTENTS

TABLE OF FIGURES

ACKNOWLEDGEMENTS

First and foremost I would like to thank Niels Jørgen Christensen for all the tutelage and advice through my time at DTU. I have really enjoyed learning computer graphics from him, and would not have made it here, were it not for him.

And thank you to Simon Tobiasen, Mikkel Hempel and Anders Rong for great friendship and mutual help and support through all the graphics courses at DTU. It has been an absolute pleasure.

Also a thank you to all the teachers I have had in all the different computer graphics related courses. Thank you to Andreas Bærentzen, Jeppe Revall Frisvad, Bent Dalgaard Larsen and Anders Wang Kristensen.

Last but definitely not least, my thanks go to my wife, Bettina. Her continuous belief in me and support through the process has been astonishing and I can truly say I would not have made it without her.

1 Introduction

As the performance of graphics hardware and the quality and speed of real-time rendering algorithms increases, the quality of real-time computer graphics is increasing exponentially. More and more lighting effects, that used to be limited to movies and still images, are making their way into games and other interactive applications.

Lately, a lot of research has been going into adding fully dynamic indirect illumination. It has even gone so far as to being defined as 'global illumination' many places. While it is a very important effect when reaching for real-time photo realism, it is certainly not the only global illumination effect currently lacking.

Another important visual effect with regard to global illumination, is volumetric scattering effects occurring when light travels through participating media. Movies and TV-shows use smoke, thick fog, dust and deep water to great extent, when trying to achieve certain moods: Ancient churches and abandoned warehouses only lit by single beams of sunlight streaming in through stained glass windows; the silhouette of a guard outlined by the light flooding the dark, dusty cellar; the murky bottle-green depths the submarine sees when searching a sunken ship deep underwater.

As in movies, the effects can be used in much the same ways in video games, and are therefore very interesting, when trying to improve the graphical fidelity and overall mood of a game. This makes it a good case area to analyze and incorporate in the context of a modern 3D graphics engine.

Light scattering is also a necessity when rendering human skin. Light passes through the top layers of the skin, bounces around, and exits. This gives skin an inherently soft look, which cannot truly be reproduced without taking sub-surface scattering into account. Most current and future high-end video games have many human characters in them, thus making rendering skin efficiently and realistically a very hot research area.

Other materials, such as marble, jade or milk also require sub-surface scattering effects to be rendered believably. Light passes through objects of these materials, but not before being scattered heavily. Light is scattered differently according to its wavelength, and colors bleed around the point of entry.

1.1 Focus of the thesis

This thesis aims to analyze and evaluate different real-time approaches to important aspects of light scattering effects. Based on the analysis, dynamic interactive methods will be implemented that combine speed and image quality to allow for the use of scattering effects in the context of a modern graphics engine.

1.1.1 The following areas will be covered

Participating media rendering enables effects like Crepuscular rays, or 'God rays'. These appear in many different situations, and are a powerful visual tool when trying to create an engaging and evocative atmosphere in a game.

For a long time, graphics hardware has been able to render distance fog. This is the effect seen when viewing objects at great distances, and is caused by light traveling through the atmosphere. This works well enough for fading out distant objects, but the simplicity of the model renders it useless for more complex foggy scenes. Analytical approaches have been made that improve on the lack of light scattering in the hardware solution. While these are simple and fast, they lack the shadows cast by objects in the path of a light beam, that are very important when creating more complex effects like god rays.

One very common way of solving differential equations on a computer is using numerical integration. In the case of the radiative transfer equation, this translates to tracing light rays through the medium, taking small steps along the ray (ray-marching). Currently most graphics engines implement some sort of deferred lighting scheme, which gives easy access to scene depth. This, together with the use of geometric objects for lighting calculations, makes ray-marching an obvious and preferable solution.

The biggest focus of current research in the field of real-time crepuscular rays is optimizing the number of samples, and the distribution of these samples to focus computations in areas where the lighting varies the most. Methods using shadow volumes, interleaved sampling and the theory of epipolar geometry will be discussed.

The work presented in this report is focused on analyzing the aforementioned ray-marching solutions, and combining their strengths to give the best results with respect to image quality and rendering speed.

Subsurface scattering and especially **skin rendering** is becoming an increasingly important research area in real-time graphics. As games move closer and closer towards movies in production values and realism, the need for rendering believable humans is increasing rapidly. Modeling light behavior as it enters and exits the skin is a widely studied topic, and computer generated characters in games are rapidly becoming more and more lifelike.

The physics of subsurface scattering makes it a complex and computationally expensive effect to simulate, but different approaches aim to simplify the solution, making it feasible to render in real-time.

Different approaches to participating media rendering and subsurface scattering will be implemented in this thesis, in the context of a modern graphics engine with the goal of allowing these effects in current generation computer games (i.e. games for Xbox 360, PlayStation 3 and high-end PCs).

1.1.2 RESTRICTIONS

When including multiple scattering, the radiative transfer equation is an integro-differential equation. The most common way of solving this type of equation on a computer, is by using Monte Carlo methods. These methods are very time consuming, and have yet to be feasible in a real-time context. If multiple scattering is ignored, the equation simplifies to a differential equation which in turn is a much more realistic target for real-time rendering. Therefore, only single scattering media are considered for this project. This will be discussed further later in the report, but is a limitation in the presented implementations.

This project will not cover the rendering of heterogeneous and emitting media (i.e. smoke or fire). These effects are closely related to efficient solving of the Navier-Stokes equations for fluid dynamics (like in

[Crane et al. 2007]), which is an area large enough to deserve a complete project of its own, and is beyond the scope of this thesis. Simple simulation of heterogeneous media using Perlin noise will be presented.

1.2 REPORT STRUCTURE

Chapter 2 will introduce the theory behind light scattering and discuss the physics and mathematics needed to render the aforementioned effects in a real-time graphics engine

Chapter 3 will present previous work in the field of volumetric light scattering with a focus on the research done in real-time methods. This includes simplifications and limitations when approaching the problem from a real-time perspective

Chapter 4 will analyze and discuss the previous methods. Based on this analysis, implementation will be described, discussing choices and limitations

Chapter 5 will show and discuss the results reached through the work done in this project

Chapter 6 will draw conclusions based on the results. This chapter also includes thoughts on improvements and future work

2 Background

This chapter covers the background theory of different basic concepts of computer graphics. Different mathematical aspects of light scattering are also covered.

2.1 Surface reflectance

When rendering realistic images, one very important aspect is how light is reflected off different materials. Some materials like polished metal and glossy plastic reflect light very well, while materials like rough wood and fabrics look very matte because of their limited reflective attributes.

When light hits an object, it does not simply bounce back off - it enters the object; bounces around a bit; some of it might be absorbed, and then exits again. This interaction is what gives materials their colors. When white light hits a blue sweater, what actually happens is, that the light enters the fabric; the fabric absorbs all non-blue wavelengths of the light, and blue light exits the sweater again giving it the blue appearance.

Describing this interaction mathematically is quite complex. The Bidirectional Surface Scattering Reflectance Distribution Function, or BSSRDF, was developed to describe light transport between any two rays hitting a given surface. The complexity of this, made it necessary to find a simpler model for surface reflection.

The concept of a Bidirectional Reflectance Distribution Function, BRDF, was introduced in [Nicodemus et al. 1977] as an approximation to the full BSSRDF where subsurface scattering is ignored. This makes it possible to calculate a realistic light-surface interaction much faster, but the lack of subsurface scattering has inherent limitations. A BRDF assumes that light enters and exits the same spot on a surface, which in most cases is incorrect, and makes it impossible to model materials such as jade or milk, but is less important for other materials like metal or plastic. A comparison of light-surface interaction using BRDFs and BSSRDFs can be seen in Figure 1.

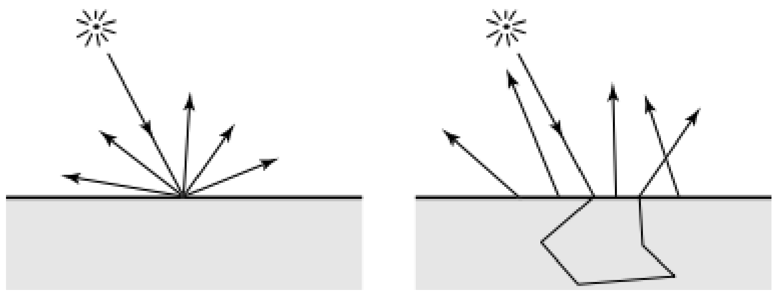


FIGURE 1 – LIGHT-SURFACE INTERACTION IN BRDF (LEFT) AND BSSRDF (RIGHT). IMAGE FROM [JENSEN ET AL. 2001]

2.1.1 Commonly used BRDFs

There exist a number of different BRDFs, each with their own strengths and weaknesses. The most common way of simulating the subsurface interaction found in a full BSSRDF is by using a Lambertian component.

This component is a pure diffuse reflective BRDF (Figure 2 – left) and can be described by the following formula:

$$f_r(x, \vec{\omega}_o, \vec{\omega}_i) = k_d = \vec{n} \cdot \vec{l}$$

Lambertian reflectance, or pure diffuse reflection, is calculated as the dot product of the surface normal, $\vec{n}$, at the point that is being shaded, and a normalized vector, $\vec{l}$, from the point in the direction of the light source. This model is used in most computer games for calculating the diffuse term of other BRDFs.

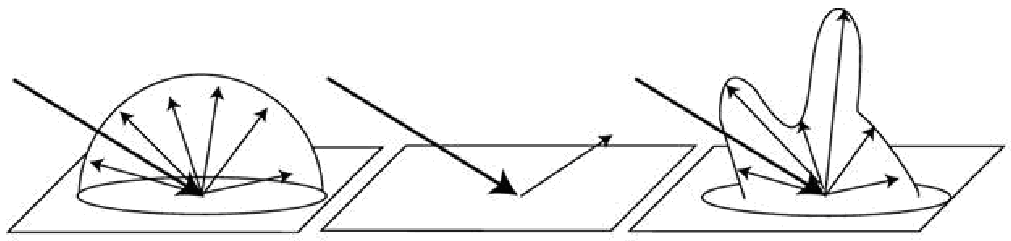


FIGURE 2 - ILLUSTRATION OF DIFFERENT BRDFS. FROM LEFT TO RIGHT: PURE DIFFUSE REFLECTION; PURE SPECULAR REFLECTION AND GLOSSY REFLECTION. IMAGE FROM [WEB - LEBEDEV]

One very commonly used model is the Phong BRDF. This model uses the diffuse reflectance from the Lambert BRDF but adds specular reflection. This part is calculated using the reflected vector of the incoming eye-vector reflected around the surface normal:

$$f_r(x, \vec{\omega}_o, \vec{\omega}_i) = k_s \frac{(\vec{r} \cdot \vec{\omega}_o)^s}{\vec{n} \cdot \vec{\omega}_i} + k_d$$

To speed up the computation of the specular term, [Blinn 1977] developed what is known as the Blinn-Phong model. This is the most common specular BRDF in use in video games today, as more physically based solutions are still very computationally heavy. The equation is:

$$f_r(x, \vec{\omega}_o, \vec{\omega}_i) = k_s \frac{\left(\vec{n} \cdot \vec{h}\right)^s}{\vec{n} \cdot \vec{\omega}_i} + k_d$$

The Blinn-Phong BRDF uses the half-vector $\vec{h}$ between the light-vector and the eye-vector, and is especially fast when using a directional light source, since the half-vector will be the same for all surfaces, as it is independent of surface curvature.

Materials like glass or water, where light passes through, need yet another model to describe the interaction with light. For this purpose, the BTDF and BSDF were developed. BTDF functions are like BRDF, but is for light exiting an object, and BSDF refers to the combination of BRDF and BTDF (see Figure 3)

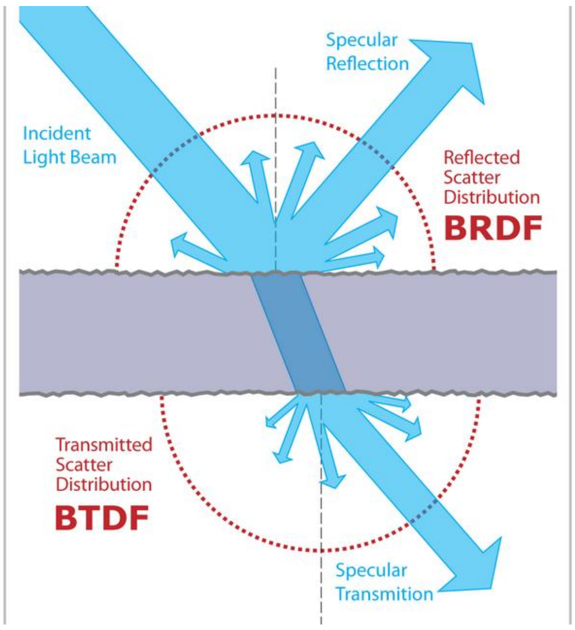


FIGURE 3 - ILLUSTRATION OF BSDF BEING THE COMBINATION OF BRDF AND BTDF. IMAGE FROM [WIKIPEDIA - BSDF]

2.1.2 Fresnel

When trying to simulate specular materials like metallic surfaces or glass, the reflected light depends very much on the incident viewing angle. Therefore the Fresnel equations are needed, which formulate the amount of reflecting light based on viewing angle. These equations are quite complex, and Christophe Schlick's approximation is used almost entirely in computer graphics. It is formulated as:

$$R(\theta) = F_0 + (1 - F_0)(1 - \cos(\theta))^5$$

Where θ is half the angle between the incoming and outgoing light vectors, and F_0 is the Fresnel term at $\theta = 0$. This approximation serves most purposes just fine, and is allowing for Fresnel reflectance to be integrated with more and more surface models. The following diagram shows how light is reflected of different materials based on incident angle.

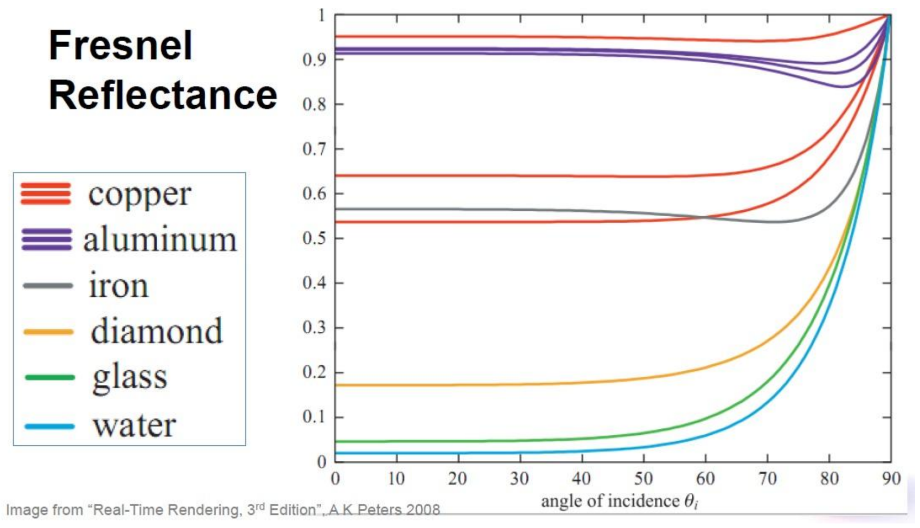


FIGURE 4 - FRESNEL REFLECTANCE FOR DIFFERENT MATERIALS AS A FUNCTION OF THE VIEWING ANGLE. IMAGE FROM [AKENINE-MÖLLER ET AL. 2008]

While Fresnel was originally only used for metallic surfaces, it is now being used increasingly in computer games in physically based BRDFs for most materials. John Hable shows how specular most materials actually are in [Web - Hable] and how the reflectance change with the viewing angle. The specularity of an otherwise very lambertian brick is illustrated below.

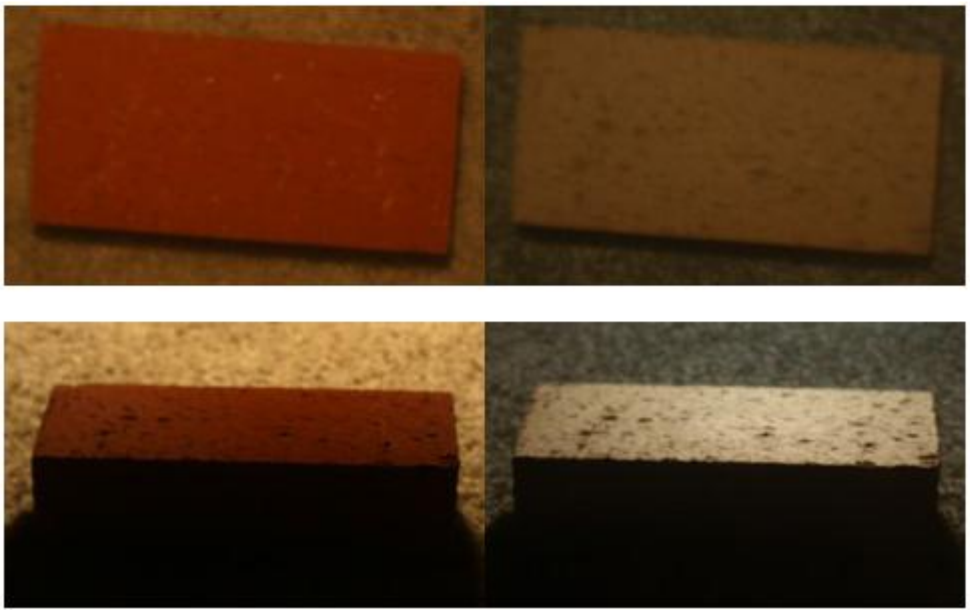


FIGURE 5 - IMAGES OF A BRICK TAKEN AT DIFFERENT ANGLES, SPLIT INTO DIFFUSE AND SPECULAR COMPONENTS. IMAGE FROM [WEB - HABLE]

2.2 THE RENDERING EQUATION

The basic idea for the rendering equation is that outgoing light at any point can be calculated as the sum of emitted and reflected light:

$$L_o(x,\vec{\omega}) = L_e(x,\vec{\omega}) + L_r(x,\vec{\omega})$$

Emitted light includes radiance emitted from fluorescent objects like light bulbs, or phosphorescent materials like glow-in-the-dark toys. Surface properties of a material determine the amount and color of reflected light as described by the BRDF.

[Kajiya 1986] formalized this into what is called the rendering equation:

$$L_o(x,\vec{\omega}_o) = L_e(x,\vec{\omega}_o) + \int_\Omega f_r(x,\vec{\omega}_i,\vec{\omega}_o)L_i(x,\vec{\omega}_i)\,(\vec{\omega}_o \cdot \vec{n})\,d\vec{\omega}_i$$

Where L_o is the outgoing light from the point x in the direction $\vec{\omega}_o$ and L_e is any emitted light. The integral over the hemisphere around the surface normal, $\vec{n}$, sums up all incoming light, and the BRDF, f_r, calculates the amount of that incoming light which is reflected. The $\vec{\omega}_o \cdot \vec{n}$ term attenuates the incoming light with regard to the outgoing (viewing) direction. The elements of the rendering equation are illustrated below.

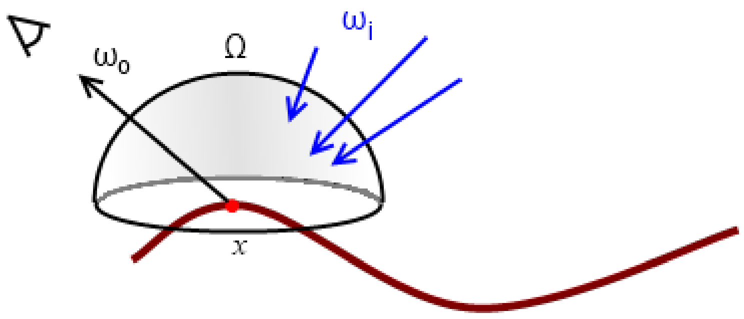


FIGURE 6 - ILLUSTRATION OF THE RENDERING EQUATION

The rendering equation in itself is not that complex, but the complexity grows when the incoming light increases. Correct multiple light bounces and indirect illumination is still reserved for offline rendering although many real-time approaches include some form of pre-computed lighting simulating the more complex lighting scenarios.

2.3 LOCAL VS. GLOBAL ILLUMINATION

In computer graphics you generally distinguish between local and global illumination effects. Local illumination refers to any lighting calculations done on a single point, without additional knowledge of the surrounding geometry. Local illumination primarily covers the shading of materials using BRDFs. No matter how advanced the rendering model is otherwise, it looks bad if the light-material interaction is off.

Any lighting effects that require more scene knowledge than what can be found in a single point, are known as global illumination effects. These include shadows cast by other objects in the scene, light reflecting off a surface before hitting the point in consideration, light focused through a refractive object causing caustics.

2.4 THE RADIATIVE TRANSFER EQUATION

One limitation of the rendering equation is, that it describes light transport in a vacuum. In reality, this is never the case, but the scattering caused by air is minimal enough for it to be ignored. When moving toward more physically correct renderings, and when trying to render light in media other than thin air, the limitations are not acceptable. Therefore we require a more complex model for these scenarios.

The theory governing the radiation field in a participating medium was first described by Subrahmanyan Chandrasekhar in his book Radiative Transfer in 1950 [Chandrasekhar 1950]. He defines the following equation for describing the directed radiation at a single point in a medium that scatters, absorbs and emits radiation. The Radiative Transfer Equation (RTE) is given as:

$$(\vec{\omega}_o \cdot \nabla)L(x,\vec{\omega}_o) = -\sigma_t(x)L(x,\vec{\omega}_o) + \sigma_s(x)\int_{4\pi} p(x,\vec{\omega}_i,\vec{\omega}_o)L(x,\vec{\omega}_i)d\vec{\omega}_i + L_e(x,\vec{\omega}_o)$$

Where $L(x,\vec{\omega}_o)$ is the radiance at the point x in the direction $\vec{\omega}_o$. The aforementioned scattering and absorption is handled by the scattering coefficient σ_s, absorption coefficient σ_a and extinction coefficient $\sigma_t = \sigma_s + \sigma_a$. L_e is any emitted radiance and the phase function p describes how the scattered radiation is distributed.

As mentioned in section 1.1.2, this thesis does not cover emissive media, which means that $L_e = 0$ and can be removed from the equation.

Parameterizing the RTE denoting the distance traveled by a light ray through the medium as s' and omitting the emissive contribution in the equation, it can be rewritten to:

$$\frac{\mathrm{d}L(s')}{\mathrm{d}s'} + \sigma_t(s')L(s') = \sigma_s(s')\int_{4\pi} p(s',\vec{\omega}',\vec{\omega})L(s',\vec{\omega}')\mathrm{d}\omega'$$

Where $\vec{\omega}'$ is the vector describing the path at the distance s'. This linear first-order differential equation can be solved by using an integration factor. In this case, we use:

$$T_r(s',s) = \exp\left(-\int_{-s}^{s} \sigma_t(t)dt\right)$$

The integral in the integration factor is known as the optical thickness, τ:

$$\tau(s',s) = \int_{s'}^{s} \sigma_t(t)dt$$

Using this integration factor, the equation can be written as follows:

$$\frac{\mathrm{d}}{\mathrm{d}s'}(T_r(s',s)L(s')) = T_r(s',s)\sigma_s(s')\int_{4\pi} p(s',\vec{\omega}',\vec{\omega})L(s',\vec{\omega}')d\vec{\omega}'$$

The next step is to integrate for $s' = 0$ to s which gives the following equation:

$$L(s) = T_r(0,s)L(0) + \int_0^s T_r(s',s)\sigma_s(s')\int_{4\pi} p(s',\vec{\omega}',\vec{\omega})L(s',\vec{\omega}')d\vec{\omega}'ds'$$

The first term on the right hand side of this equation is called the direct transmission term. For homogeneous media, the optical thickness $\tau(0,s) = \sigma_t s$ making the direct transmission:

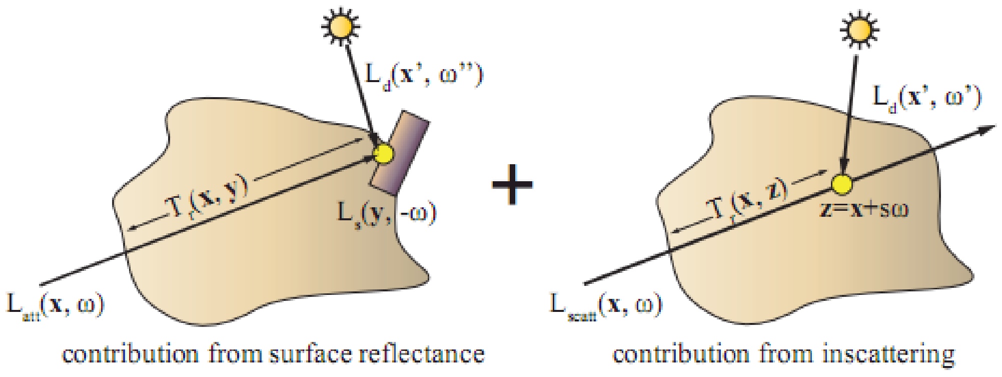
$$T_r(0,s) = e^{-\sigma_t s}$$

,which is easily calculable.

As mentioned in section 1.1.2, the solution presented in this thesis limits itself to single scattering. This simplifies the equation further, making it possible to solve the second term of the right hand side of the equation by use of a finite Riemann summation. The resulting equation is:

$$L(s) \approx L(0)e^{-\sigma_t s} + \sum_{n=0}^{N} \sigma_s \frac{\Phi}{d^2} \cdot e^{-\sigma_t \cdot d} \cdot p(s', \vec{\omega}', \vec{\omega}) \cdot e^{-\sigma_t \cdot s'} \Delta l$$

Where d is the distance from the current point to the light source, Φ is the intensity of the light source and $s' = n \cdot \Delta l$ is the distance to the current point from the camera. $L(0)$ is the light intensity at the distance s, which is the intensity of the light source attenuated by the optical thickness from the light source to the point. The algorithm is illustrated in Figure 7 although notation is a bit different.



$L_d(\mathbf{x}', \omega'')$
$T_r(\mathbf{x}, \mathbf{y})$
$L_s(\mathbf{y}, -\omega)$
$L_{att}(\mathbf{x}, \omega)$
contribution from surface reflectance
+
$L_d(\mathbf{x}', \omega')$
$T_r(\mathbf{x}, \mathbf{z})$
$\mathbf{z}=\mathbf{x}+s\omega$
$L_{scatt}(\mathbf{x}, \omega)$
contribution from inscattering

FIGURE 7 - SINGLE SCATTERING CALCULATION. IMAGE FROM [ENGELHARDT AND DACHSBACHER 2010]

2.5 Skin Rendering

Human skin has always been one of the most difficult materials to render. We as humans are very sensitive to the appearance of skin, and even close approximations will look very wrong to most people until they are very realistic. On the surface, skin has thousands of small pores and wrinkles, which are noticeable immediately if missing. Light is scattered under the surface of the skin in different amounts for different wavelengths of light, giving it a very smooth look. Therefore, more advanced specular and diffuse reflection models are needed. The following figure illustrates the multilayered skin model:

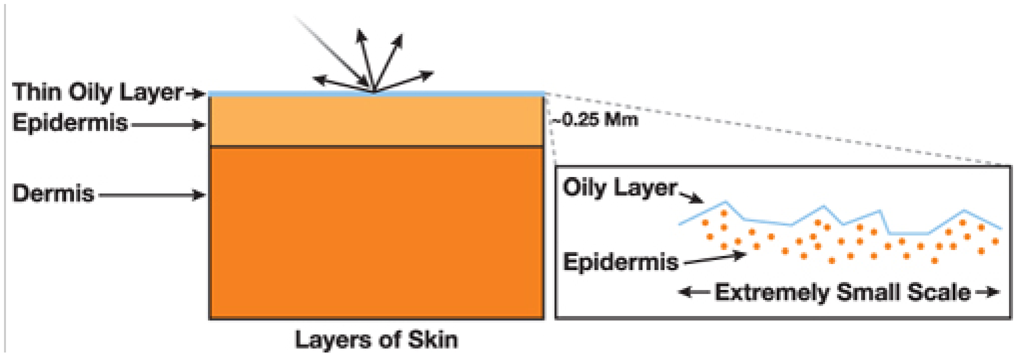


FIGURE 8 - THE STRUCTURE OF SKIN AS A MULTI-LAYERED MODEL. IMAGE FROM [D'EON AND LUEBKE 2007]

The thin oily layer and the epidermis do reflect an amount of light directly off the surface without coloring it. This specular reflection is more complex than can be modeled by the Blinn-Phong BRDF, and a more advanced, physically based BRDF is required. [Kelemen and Szirmay-Kalos 2001] presented a BRDF that has been shown to give very realistic specular reflections for skin rendering.

The light not reflected off the surface of the skin, penetrates the top-most layers, and is scattered within the dermis before exiting again (see Figure 9). This scattering cannot be simulated by a simple Lambertian diffuse model which renders the skin looking dry an unrealistic.

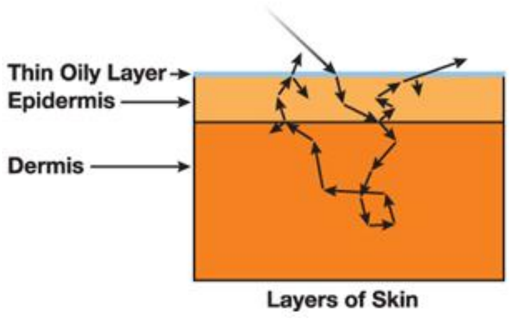


FIGURE 9 - THE SUBSURFACE SCATTERING OF SKIN. IMAGE FROM [D'EON AND LUEBKE 2007]

One aspect of skin, is that light entering the lower layers, quickly becomes very diffuse and scattered in all directions. This means that light doesn't travel very far through skin, and this is a property that will be exploited in the implementation described in section 4.3.1.

3 PREVIOUS WORK

Since scattering effects add a lot to computer generated images, a lot of research and work has been made in the field. Both offline and in real-time, scattering effects add a lot to an image, and is an important global illumination effect. This section of the thesis will go through the different approaches to rendering scattering media.

Another very important aspect of global illumination is shadows, and shadows are also needed for realistic rendering of scattering media.

3.1 SHADOWS

When trying to visually place an object in space in an image, one very important visual clue is where the object casts a shadow. It can be very difficult to see if an object is floating or stationary without a guiding shadow as shown in Figure 10. The same objects are rendered, but with different shadows, illustrating how much of a difference it makes. Shadows are among the very first global illumination effects to be rendered in computer graphics, and many different approaches have been made in both offline and real-time rendering.

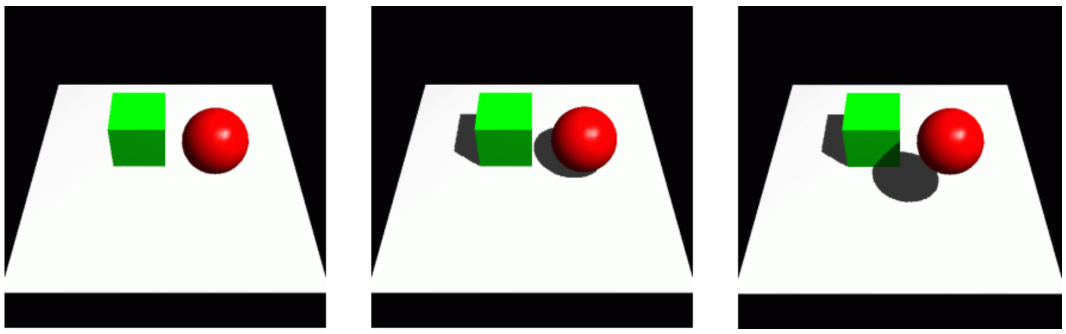


FIGURE 10 - USING SHADOWS TO VISUALLY PLACE OBJECTS IN SPACE (FROM [FREIBURG 2007])

In ray tracing, shadows can be implemented quite easily by casting additional rays from intersected objects. These shadow rays are cast towards all light sources, and tell whether or not a point is lit by that light source or in shadow, if the ray hits an object before reaching the light source. When using area light sources, these have an expansion, and therefore multiple shadow rays are needed. This gives realistic soft shadows with both umbra and penumbra as illustrated in Figure 11 and is highly sought after in real-time rendering.

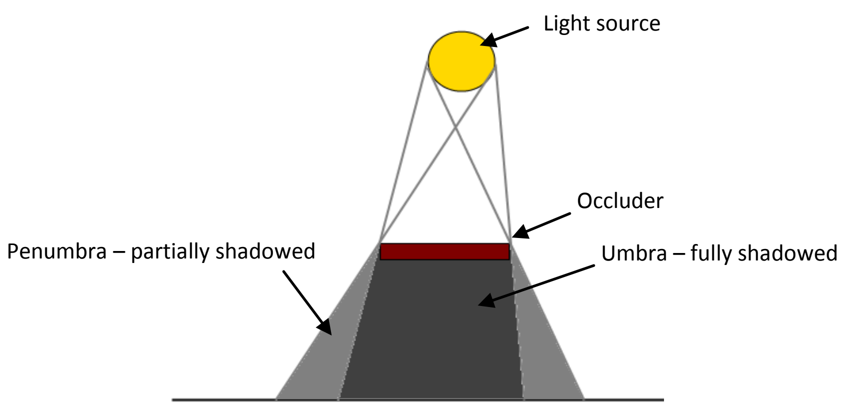


FIGURE 11 - SHADOW FROM AN AREA LIGHT SOURCE

Correct soft shadows are the holy grail of real-time shadow rendering. The problem exists because the area light source has not yet found its way to real-time rendering. When doing shading on a graphics card, you generally can only have one direction from a scene point to a light source. This means a vector from the scene point to either a point light source or spot light, or a general direction for directional light sources (usually the sun). Also there is no additional scene information available.

The most common way of including shadows in real-time graphics, is by way of a Shadow Map. The idea is to first render the scene as seen from the light source, storing the depth buffer in a render target. After that, the point of view is switched to the the camera, and the scene is rendered again, this time looking every point up in the shadow map to see if the light source has seen something closer to itself than the point in question. If this is the case, the given point must be in shadow. The technique is illustrated in Figure 12.

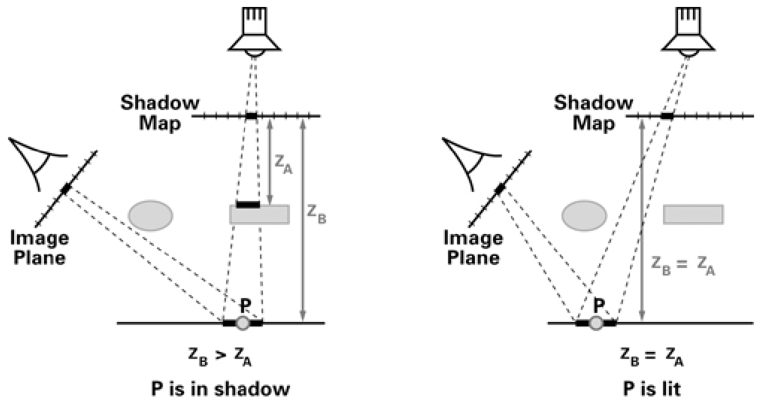


FIGURE 12 - HOW SHADOW MAPPING WORKS (PICTURE FROM THE CG TUTORIAL BOOK, CHAPTER 9)

Many different approaches have been made toward softening the edges of shadows in real-time rendering. Percentage-closer filtering (PCF) works by sampling the shadow map several times around the point being calculated, and averaging over the visibility of all nearby texels. This gives a decreasing shadow value closer to the edge of a cast shadow. This is a very easy method, but it lacks the sharpening of shadow edges that appear close to the shadow casting object.

3.2 Participating media

In offline rendering, participating media rendering has been studied for a long time. [Pharr and Humphreys 2004] describe a solution based on ray-marching within the scattering medium, using Monte Carlo techniques for modeling the scattering of particles. Renderings of mist and water using path tracing can be seen in Figure 13. Path tracing through volumetric scattering media is extremely time consuming, and converges very slowly to a visually pleasant solution. Images need very long computation times, if you wish to avoid grainy and noise-filled results. [Jensen and Christensen 1998] developed the very commonly used method of volumetric photon maps, which greatly speeds up rendering time when ray tracing through participating media. This method has been improved in [Jarosz et al. 2008] and there is much research in the area, as it is still a slow and noise-prone aspect of rendering.

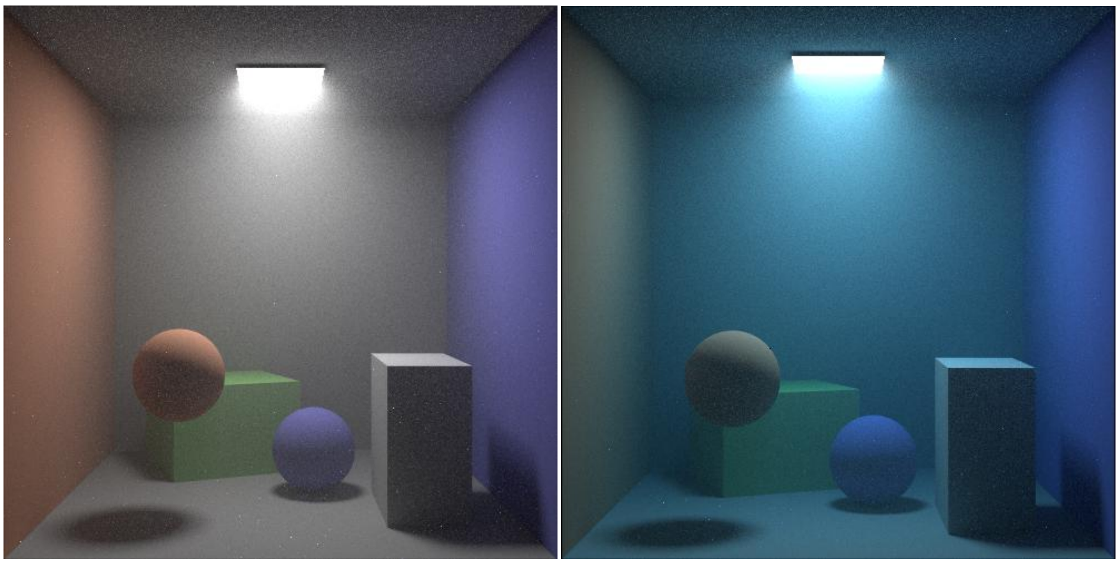


FIGURE 13 - SCATTERING FROM MIST AND WATER RENDERED USING PATH TRACING

These offline methods are not nearly fast enough when moving to the realm of real-time graphics, but there have been different approaches to the rendering of participating media in real-time. Most of them limit themselves to the rendering of single scattering, as this gives convincing results in most cases, while still being plausible in real-time.

[Sun et al. 2005] developed a particularly fast analytical solution which allows for realistic light scattering in foggy or hazy scenes. This model is simple and fast for effects like the corona around street lamps on a foggy night (Figure 14), but has inherent limitations in not taking visibility into account. Their solution does include a good scattering model, which will be discussed later.

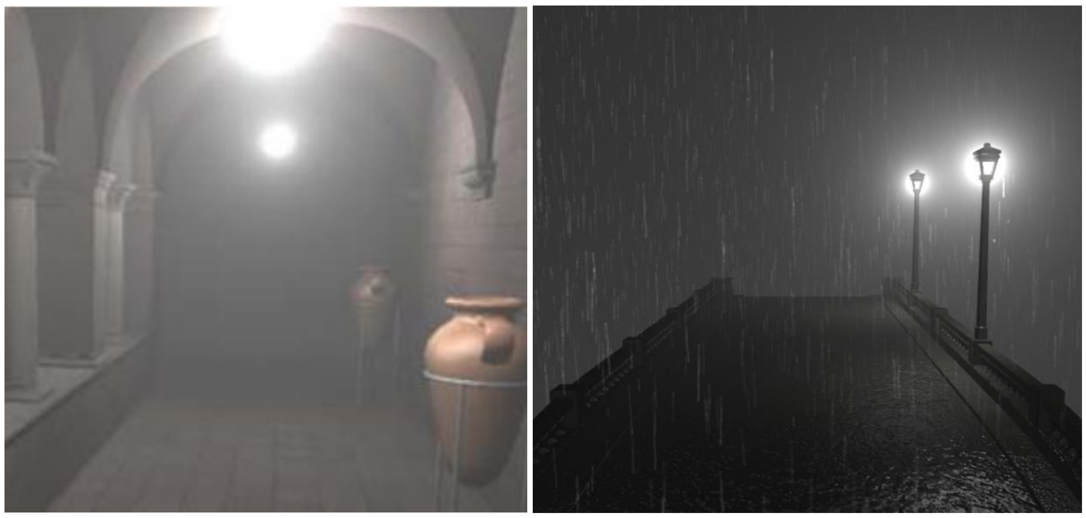


FIGURE 14 - EXAMPLES OF SCATTERING FROM [SUN ET AL. 2005] (LEFT) AND FROM RAIN-SAMPLE BY NVIDIA (RIGHT)

The most visually important features of participating media-rendering, including light shafts, require visibility calculations. There are several general approaches to this problem, and these approaches can roughly be split into four groups:

SHADOW VOLUMES

[Biri et al. 2006], [James 2003] and [Mech 2001] all use shadow volumes to calculate shadowed areas of the participating medium. Shadow volumes are currently not used very much as they add additional geometry to the rendering, where games are already pushing the limits. On top of that, they all require some sorting of the shadow planes, making them less useful. [Billeter et al. 2010] extrude light volumes from a shadow map instead of shadow volumes, but similar limitations exist. Their algorithm is illustrated in Figure 15

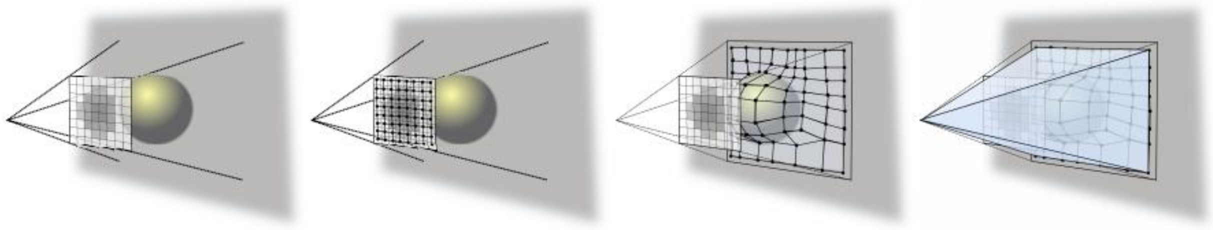


FIGURE 15 - EXTRUDING LIGHT VOLUMES FROM A SHADOW MAP. FROM [BILLETER ET AL. 2010]

Slicing techniques

[Dobashi et al. 2002] and [Mitchell 2004] have used slicing techniques known from volume rendering to render light shafts with shadows. Quads parallel to the view plane are rendered, sampling a shadow map for visibility. These quads are then rendered back to front, and blending them together additively. This approach suffers from aliasing issues, and a large number of planes are required for smooth results. This problem was addressed by [Imagire et al. 2007] where sampling is averaged over nearby slices. This reduces the number of required planes, and works as anti-aliasing of the result. This approach resembles ray-marching which will be covered shortly.An illustration of the basics of the slicing scheme can be seen in Figure 16

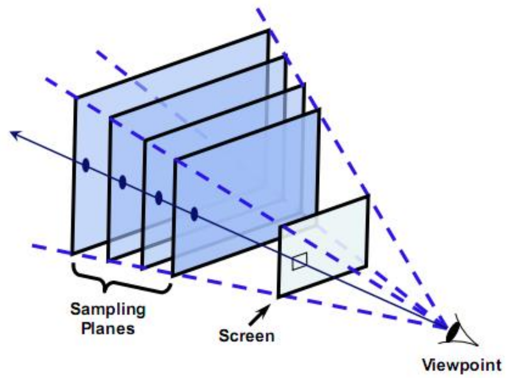


FIGURE 16 - SLICING SCHEME FROM [MITCHELL 2004]

Post-process effect

[Mitchell 2007] developed a very fast screen-space effect. It works by ray-marching in screen-space from a given point to the location of the light source. It sums up all samples where no geometry was encountered, and attenuates them, by simulating volumetric scattering, giving the effect of sun shafts (Figure 17).

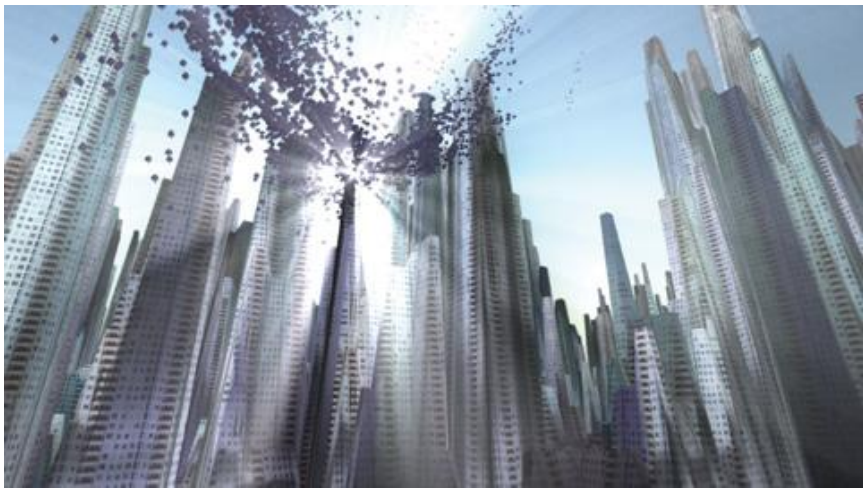


FIGURE 17 - SCREEN-SPACE SUN SHAFTS FROM [MITCHELL 2007]

[Sousa 2008] extended this method to use a radial blur, centered on the position of the sun in screen-space coordinates, on the depth buffer as a mask for the God rays (Figure 18). This works very well with deferred rendering approaches where a depth buffer is readily available.

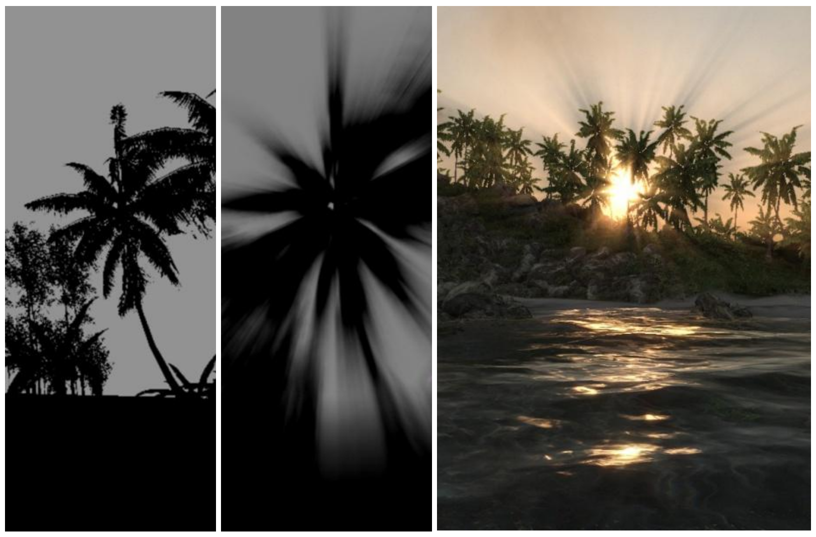


FIGURE 18 - RADIAL BLUR USED AS A MASK FOR SCREEN-SPACE GOD RAYS (FROM [SOUSA 2008])

These methods allow for believable shadows from geometry, but their simplicity means they are quite error-prone. As all geometry has the same effect on the shadows, objects behind the light source as well as objects very close to the camera will affect shadows as much as 'correct' shadow casters, which can be undesired. The effect works well as long as the light source (most often the sun) is visible on screen. Regardless of its shortcomings, it is a very useful approach for large outdoor scenes, where light sources are very far away (like the sun in Figure 18). Here the speed it offers, as compared to ray-marching solutions, easily makes up for the limitations.

RAY-MARCHING

Solutions based on ray-marching have become the most common. They are straight forward to implement, and most current research go into optimizing the required number of samples and the location of these samples.

Ray marching is done as a full-screen pass, where a pixel shader steps through the scene from the camera to the depth buffer, accumulating samples at every step. Each step is visibility tested using a shadow map, and calculations are made for points not in shadow (see Figure 19). Raw ray marching needs quite a lot of samples to give decent results, but many optimizations exist.

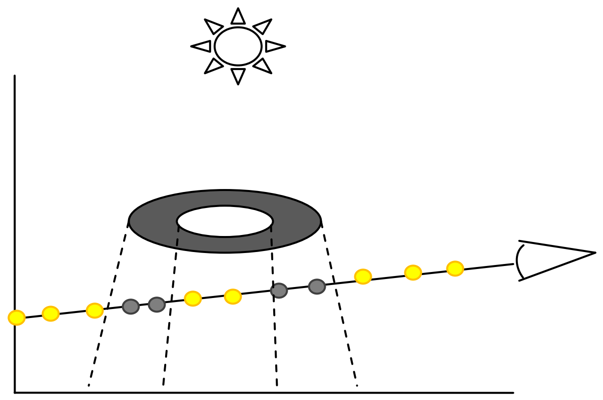


FIGURE 19 - RAY MARCHING ILLUSTRATED

[Wyman and Ramsey 2008] propose, using shadow volumes, to store depth-values for the nearest and furthest shadowed areas, ray-marching only between these planes. This approach also allows them to find pixels that don't encounter any shadows, and use the analytical approach from [Sun et al. 2005] without needing to ray-march. They also include the use of textured spot lights, but in that case, their optimizations are less useful. The technique and results are shown in Figure 20.

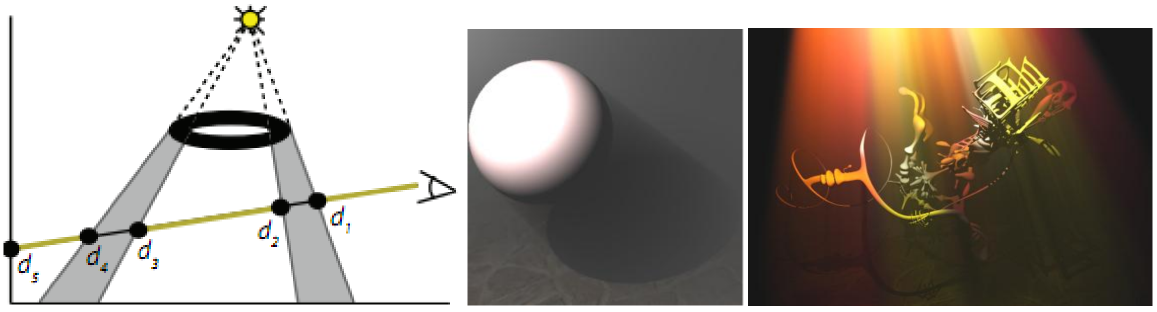


FIGURE 20 - TECHNIQUE AND RESULTS FROM [WYMAN AND RAMSEY 2008]

When sampling textured spot lights, regular ray-marching in view-space or world-space can have problems retaining details of the projected image. [Gautron et al. 2009] have addressed this issue by doing their sampling in shadow map-space, evenly spacing their samples over the projected texture. They also include an optimization that is linked to deferred lighting where spot lights are often rendered as cones, allowing them to find entry and exit points of the camera-ray with regards to shadow map-space (see Figure 21).

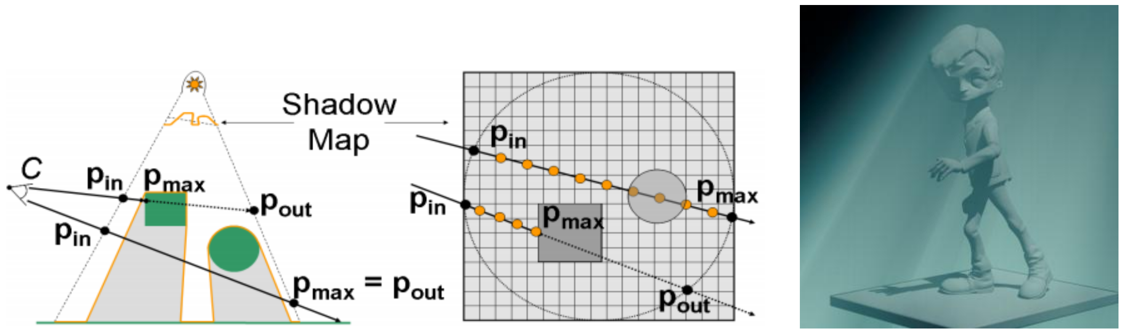


FIGURE 21 - SAMPLING PATTERN (LEFT) AND FINAL RESULT (RIGHT) FROM [GAUTRON ET AL. 2009]

[NVIDIA 2008] optimize ray-marching by creating a mipmap pyramid for the shadow map, storing min and max depths. This reduces the number of required visibility samples, and allows them to take larger steps in many cases. Sample-aliasing is reduced by jittering the starting point for each ray. This paper also includes a Sobel-filter approach to up-sampling, which allows them to render the volumetric lighting at a lower resolution, while still keeping smooth edges (see Figure 22). Like NVIDIA, [Tóth and Umenhoffer 2009] also jitter the samples. They use interleaved sampling ([Keller and Heidrich 2001]) in screen-space and average over neighboring pixels.

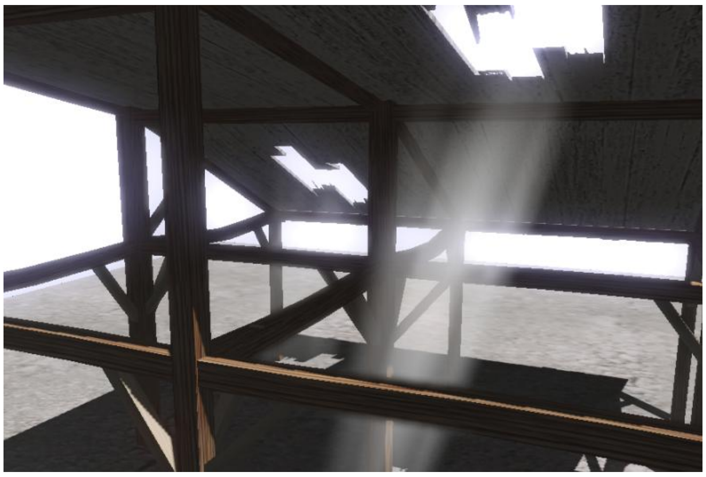


FIGURE 22 - RESULT FROM [NVIDIA 2008]

[Engelhardt and Dachsbacher 2010] observed that inscattered light varies smoothly (most of the time) along the light shafts. They use the theory of epipolar geometry to focus the samples on lines emanating from the light source in screen-space, and smoothly interpolate between these samples along the epipolar lines. Depth discontinuities are found and sampling is focused around these points. Figure 23 shows how this sampling pattern leads to very crisp light shafts.

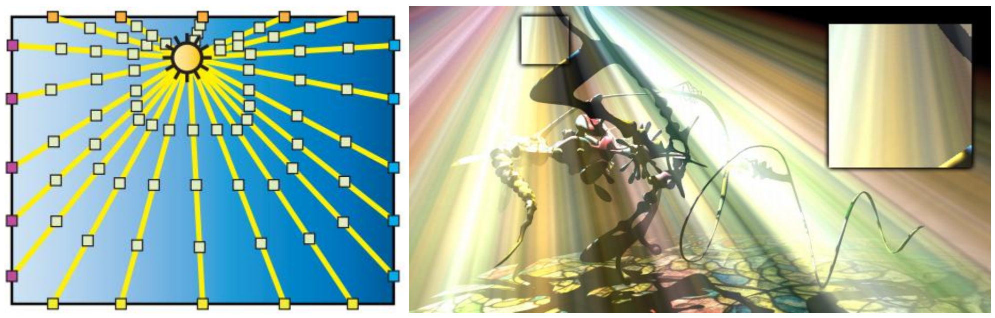


FIGURE 23 - CRISP SHADOW BOUNDARIES USING EPIPOLAR SAMPLING

The theory of epipolar geometry lends itself very nicely to rendering of participating media. [Wyman 2011] creates voxelized shadow volumes in epipolar space, and [Chen et al. 2011] create a 1D heightmap in epipolar space, they traverse to find shadowed areas.

Lately, [Kaplanyan and Dachsbacher 2010] presented work in real-time indirect illumination where light is propagated through a voxel-grid. This has given results, where participating media rendering is easily integrated and this method has been extended to allow for interactive rendering of multiple scattering in heterogeneous media as well [Engelhardt et al. 2010]. This way of having lighting in a voxel-grid is even more integrated in the graphics engine used for the game LittleBigPlanet 2. Here the scene is 'voxelized' every frame and all lighting calculations are done by sampling the resulting grid. This allows for easy volumetric shadows, but the sharpness and quality is quite limited on current hardware [Evans and Kirczenow 2011].

3.3 SUBSURFACE SCATTERING

Estimating subsurface scattering in real-time is very complex, and the most realistic approaches are still too expensive for use in video games. Simpler methods are gaining popularity as subsurface scattering adds a lot to scenes in many scenarios.

[Dachsbacher and Stamminger 2003] expand on the idea of shadow maps to include more information. This information is then used to calculate an approximation of how much radiance has flown though the model. The same approach was made by [Chang et al. 2008] and [Ki 2009] extended this algorithm with very good results.

[Sousa 2008] use a very simple approach to scattering through leaves, where their artists paint a 'subsurface texture map' giving them a factor indicating how much light passes through the leaf at any given point. This factor is simply multiplied by double-sided lighting for a quick and dirty approach to subsurface scattering.

[Barré-Brisebois and Bouchard 2011] expand this concept to more voluminous models, by adding a texture to each model, where they have stored ambient occlusion calculations for the inside of the model. They flip all normals, and do regular ambient occlusion calculations. This gives them a rough estimate of local thickness of the model, which is then used to attenuate a translucency factor. Results from this method are shown in Figure 24

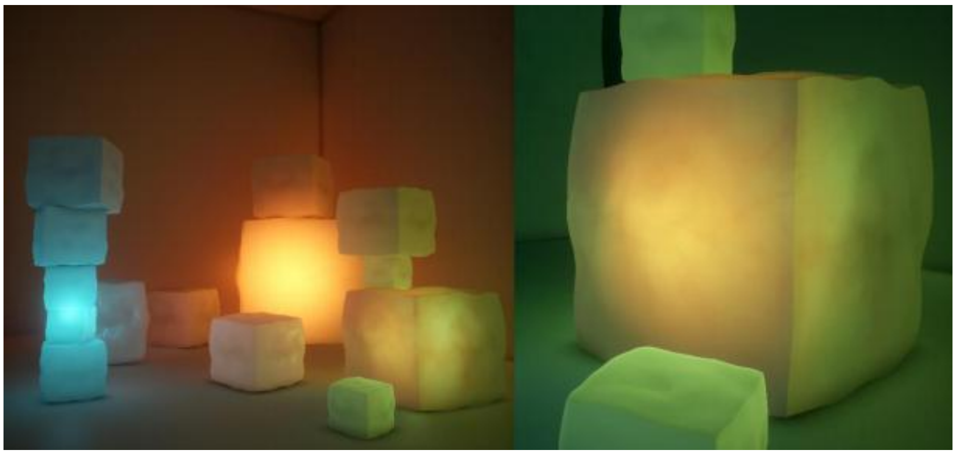


FIGURE 24 - RESULTS ACHIEVED BY [BARRÉ-BRISEBOIS AND BOUCHARD 2011]

3.3.1 SKIN RENDERING

In computer graphics, human skin is a very complex and difficult material to render. [Jensen et al. 2001] developed an approximation to full BSSRDF rendering, which enabled (and still enables) movies to include increasingly realistic computer generated humans (and human-like creatures). Before this, rendering had focused on simulating the simpler BRDF, because it was too costly to go beyond this. Offline rendering has come very far (see Figure 25), and the most realistic real-time simulations are still very costly, limiting their usability.

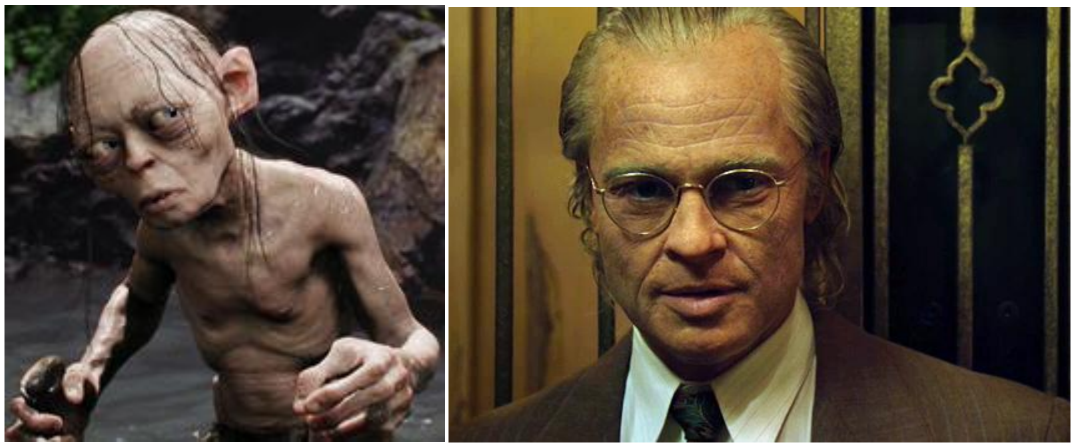


FIGURE 25 - EXAMPLES OF SKIN RENDERING FROM MOVIES (LEFT FROM [WIKIPEDIA - GOLLUM] AND RIGHT FROM [WEB - BENJAMIN BUTTON])

Based on the observation that skin has quite limited scattering distance and limited curvature, [Borshukov and Lewis 2003] developed what is known as Texture Space Diffusion. This technique works by rendering lighting into a light map in the models texture coordinates, and performed a custom blur on this texture to simulate light scattering (see Figure 26). The blur is separate in red, green and blue, simulating the difference in light scattering for different wavelengths.

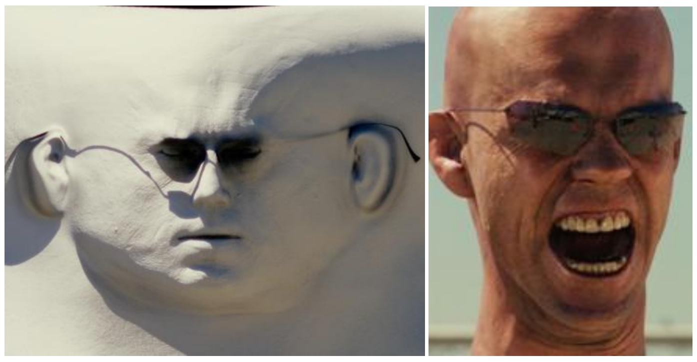


FIGURE 26 - TEXTURE SPACE DIFFUSION IN THE MATRIX RELOADED (FROM [BORSHUKOV AND LEWIS 2003])

In [Green 2004] and [Gosselin 2004] the notion of texture space diffusion was used, but with a simpler 2-pass Gaussian blur. The still gold standard in real-time skin rendering came in [d'Eon and Luebke 2007] where texture space diffusion was coupled with a sum of Gaussians approximations. These Gaussian blurs were fit from real measured data for skin and, together with a very detailed model, gave very convincing results (see Figure 27). Further details of this method will be described in section 4.3.1.

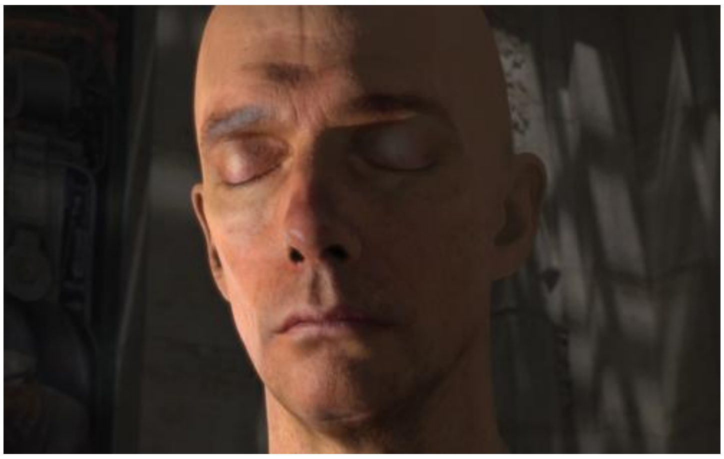


FIGURE 27 - RESULTING IMAGE FROM [D'EON AND LUEBKE 2007]

[Hable et al. 2009] kept the blurring in texture-space, but developed a custom 12-tap blur instead of the five blur-passes needed for [d'Eon and Luebke 2007]. It is not as correct, but much faster and the results are comparable. Figure 28 illustrates the different approaches. Implementation details of this method will also be covered in section 4.3.1.

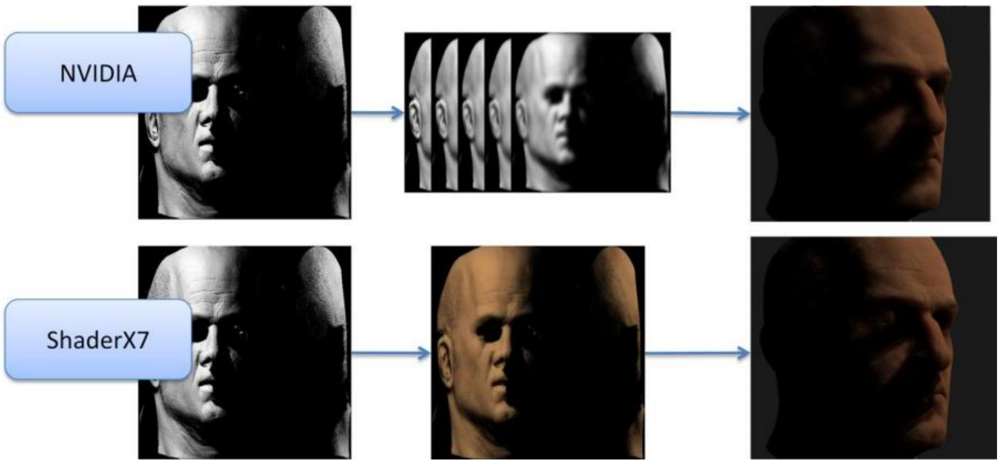


FIGURE 28 - COMPARISON OF TEXTURE SPACE DIFFUSION METHODS. TOP: 5-PASS FROM [D'EON AND LUEBKE 2007]. BOTTOM: 12-TAP SINGLE PASS FROM [HABLE ET AL. 2009] (IMAGE FROM [HABLE 2010])

While the texture-space methods work very well, they require a light map per texture, which can be expensive if multiple characters are on-screen. Another issue can arise when it is not a head but something smaller, like for example a hand. Here, textures will often not be as detailed, or could be part of a bigger texture which can lead to difficulties. [Jimenez et al. 2009] move the blur-pass to screen-space. This technique has the advantage of working equally well no matter how texture coordinates and textures are laid out and it also has a fixed cost which is more predictable. It does require a light map as seen from the camera, and therefore works best in conjunction with a light pre-pass renderer. This method has a problem in determining curvature and blurring over non-adjoining areas like from nose to cheek for example. This issue was handled in [Mikkelsen 2010] by use of a pseudo-separable cross bilateral filter, giving very decent results that are much faster than texture space diffusion. Figure 29 illustrates the difference between a regular screenspace blur, and the technique developed by [Mikkelsen 2010].

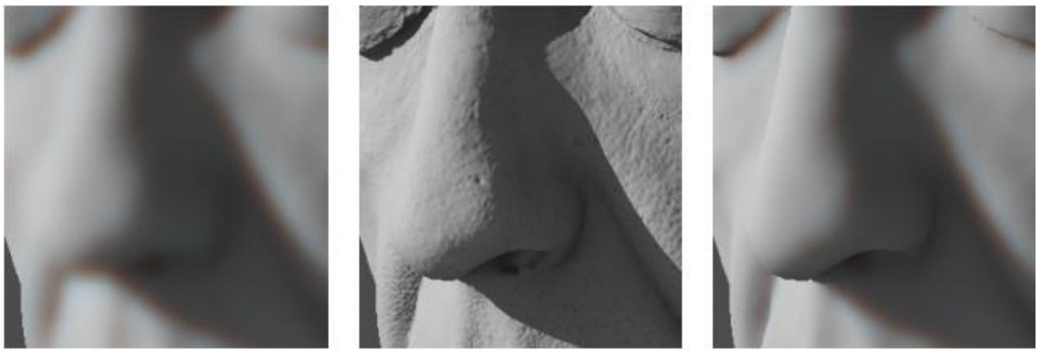


FIGURE 29 - DIFFUSE (CENTER), REGULAR 2D BLUR (LEFT) AND CBF-APPROXIMATION (RIGHT) (FIGURE FROM [MIKKELSEN 2010])

Recent work in subsurface scattering has used light propagation volumes (from [Kaplanyan and Dachsbacher 2010]) to estimate the flow of light through heterogeneous materials in real-time [Børlum et al. 2011].

4 ANALYSIS AND IMPLEMENTATION

This chapter will cover analysis and implementation decisions for each of the covered areas of scattering, and discuss any related topics such as render engine design.

4.1 DEFERRED SHADING AND THE LIGHT PRE-PASS RENDERER

As realism in real-time computer graphics improves, so does the need for more realistic scenes. One limitation that has been around for a long time is the lack of a good solution to having many lights in a scene. When looking around the real world, there is usually any number of light sources affecting what you see, and in computer generated scenes, light sources can be a powerful artistic tool when setting the mood of a scene. Additional light sources can also be used to simulate other effects, like indirect illumination and particle based effects like fire. This problem has been addressed in most modern graphics engines by some form of deferred rendering scheme.

The idea of deferred shading started with [Hargreaves 2004] as a possible solution to the many-lights problem in regular forward rendering. When rendering lit objects in the usual way, each object is rendered, applying all lights in a single shader. This works fine for scenes with very few lights (outside, lit only by sunlight for example), but as the number of lights increases, the complexity grows exponentially, and shader instruction limitations are suddenly a limitation. Deferred shading handles this by decoupling the lighting from the geometry rendering. All objects are rendered, filling render targets with scene depth, normals, diffuse color and any other material properties needed for lighting. This is called the geometry buffer or G-buffer. Afterwards, all lights are applied as 2D postprocess, reading necessary information from the render targets from the G-buffer. World-space position is computed from the stored depth, and normals and specular strength combine for shading calculations. Diffuse texture colors are read, and a final color can be output. This greatly simplifies scenarios where many lights affect scene-objects, and means that there is little or no difference in the cost of rendering many small lights or few larger.

The drawback of this method is the memory needed for multiple large render targets, and the many reads from these needed for lighting calculations. As a response to this problem, [Engel 2009] came up with the concept of a Light pre-pass renderer. Here, the geometry is first rendered to a much smaller G-buffer of only two render targets. Scene depth, normals and specular reflection is usually stored (see Figure 30). After this, the lighting is performed in the same way as in deferred shading, but without diffuse albedo. Finally the scene geometry is rendered again, this time applying the lighting from the light pass as simple screen-space look-ups in a texture.

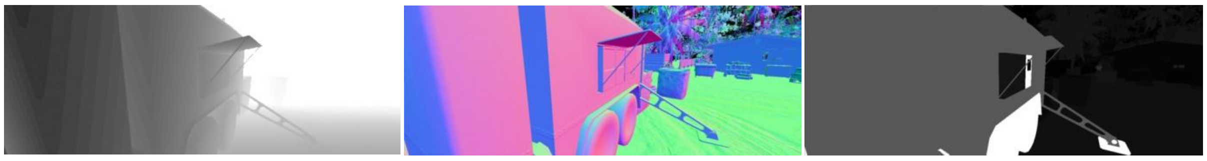


FIGURE 30 - G-BUFFER LAYOUT OF CRYENGINE 3. FROM LEFT: DEPTH, NORMALS AND SPECULAR POWER (FROM [MITTRING 2009])

When doing lighting in a deferred renderer, the number of calculations can be reduced by rendering bounding objects for each light source. For example a sphere can be used for point lights and a cone for spot lights. This means that lighting calculations are only done for pixels that could actually be reached by the light source (see Figure 31). Newer implementations like [Andersson 2011] use the Compute Shader in DirectX 11 (shader model 5.0) and tile based calculations to limit overdraw when many lights overlap in large scenes. This is a very interesting approach, but is beyond the scope of this thesis.

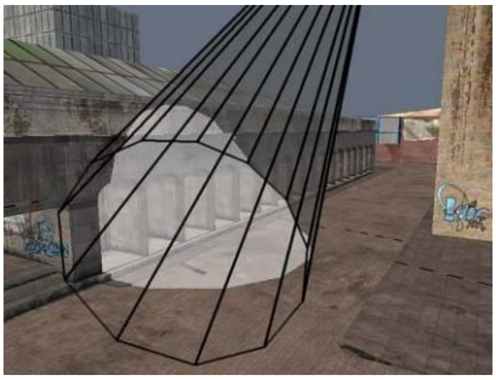


FIGURE 31 - THE USE OF A BOUNDING VOLUME FOR LIGHTING CALCULATIONS (FROM [HARGREAVES 2004])

One very useful feature about these approaches to rendering engines is that you gain access to a depth buffer in every frame. This comes in very handy when doing different forms of participating media rendering as will be covered later.

Both methods have their strengths and weaknesses, and both methods are used in high-end graphics engines. Examples include deferred shading in the Frostbite 2-engine used for Battlefield 3 ([Andersson 2011]) and Light pre-pass rendering in CryEngine 3 used for Crysis 2 ([Mittring 2009]).

As the goal of this thesis is to implement scattering effects in the context of a current game engine, I have chosen to implement a simple light pre-pass renderer. It allows for more flexibility than regular forward rendering, and serves the needs of this thesis very well.

I have chosen a simple G-buffer layout with only scene depth, and normals. This comes down to the focus of the thesis being volumetric scattering, and simple lambertian shading is sufficient to illustrate these effects, which means that other material parameters can be ignored (except for skin shading, but this will be discussed later).

4.2 PARTICIPATING MEDIA

The first implemented effect is volumetric single scattering media. This is a visually powerful effect, and one that is just beginning to enter computer games and graphics engines, as it is still computationally hard. It can really add a lot to a scene, and is therefore an interesting area to cover.

To get the best result when rendering scattering media, the analytical method proposed by [Sun et al. 2005] just doesn't cut it. The lack of volumetric shadows is very clear, and is noticed immediately by the viewer. The speed gained from using this method is noticeable enough that whenever you can guarantee no shadows will be cast, or that they will be unnoticeable, this is definitely preferable. A good example is car headlights and lamps in slightly misty or foggy weather. These lights are not powerful enough combined with the subtle nature of slight mist that they cast noticeable shadows. For light shafts coming in through a window in a dark room, it is very noticeable though. If there are no shadows in these, it can be very difficult to place them, and it simply looks wrong.

The optimal solution is obviously to use the analytical method for all pixels where there are no shadows, and only ray-march where this is necessary. [Wyman and Ramsey 2008] solve this by storing depth values for the frontmost and backmost shadow polygons, and can then check when they are not defined. This method relies on shadow volumes, which is not desirable as they are bound by geometric scene complexity, and can introduce many additional polygons to an already heavy scene.

I have chosen a ray-marching approach for my implementation, as it can give very good results, close to ground truth. Since I have chosen to use a light pre-pass renderer as the basis of my solution and use bounding cones for spot lights, the first optimization I have made is to limit sampling to within the boundaries of the light cone. This is done by first rendering front faces of the cone to a depth buffer. Then for the main ray-marching pass, back faces are rendered and the front face depth for the pixel is read from the stored buffer. This allows for tighter sampling, and no samples are wasted on areas where the light source doesn't hit anyway.

The second optimization I have chosen is to render the scattering pass to a lower resolution buffer, and then up-sampling it for the final image. This approach results in the shadow edges become a bit blurry, but this is in reality one of the effects additional effect for modeling multiple scattering as well, and is not a huge drawback. The resolution is halved in both x and y resulting in only one fourth as many pixels needing calculation. This approach requires extra up-sampling to avoid edge-aliasing as shown in Figure 32.

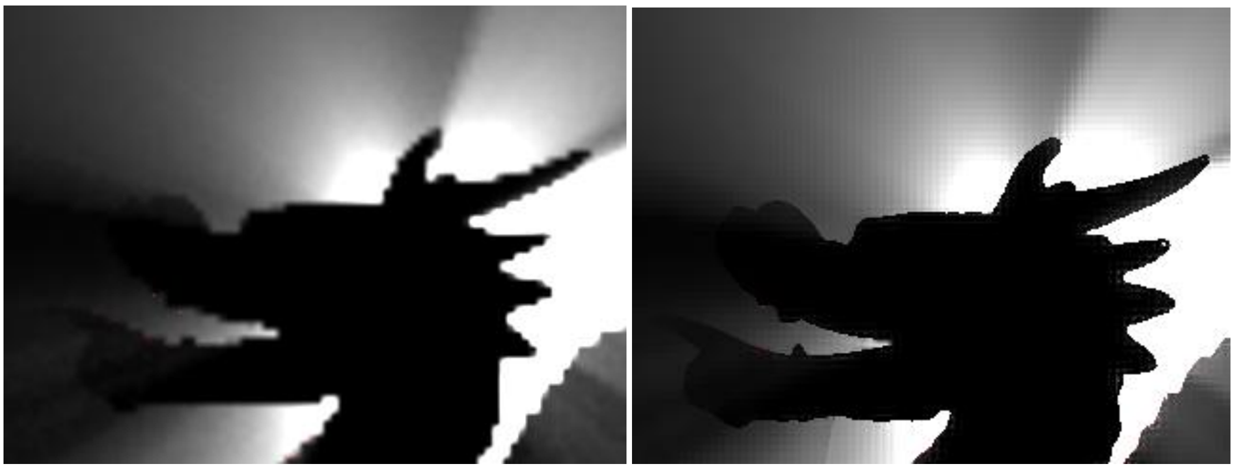


FIGURE 32 - EDGE-ALIASING CAUSED BY LOW RESOLUTION FOG CALCULATIONS WITHOUT UP-SAMPLING (LEFT) AND WITH (RIGHT)

When trying to reduce the number of required samples, one effective optimization is to jitter the starting points of the ray-marching. Jittering can either be done by interleaved sampling ([Tóth and Umenhoffer 2009]) or by sampling a random noise texture ([NVIDIA 2008]). I found that the best results are achieved by random jitter, and have implemented creation of a noise texture, and sampling of this when starting each ray-march. The up-sampling done to remove edge aliasing, also works as a blur pass, improving the jittered sampling Figure 33 illustrates what is gained by jittering.

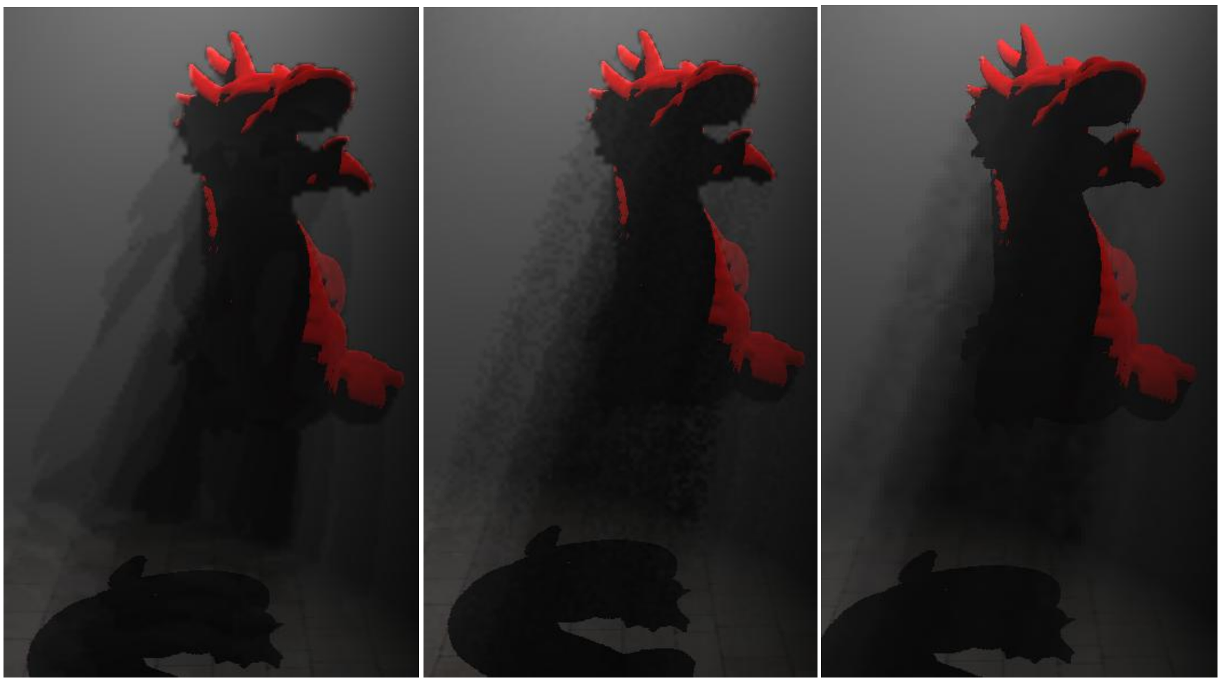


FIGURE 33 - BANDING FROM TOO FEW SAMPLES (LEFT). JITTERING ADDED (MIDDLE). NEIGHBOR SAMPLING BLUR (RIGHT)

One common technique when rendering participating media is to add a texture to the spot light to simulate a stained glass window. This texture is sampled in the same way as the shadow map is sampled, and the scattering contribution color is multiplied by the stored color. This technique can give some very pleasant effects shown in Figure 34.

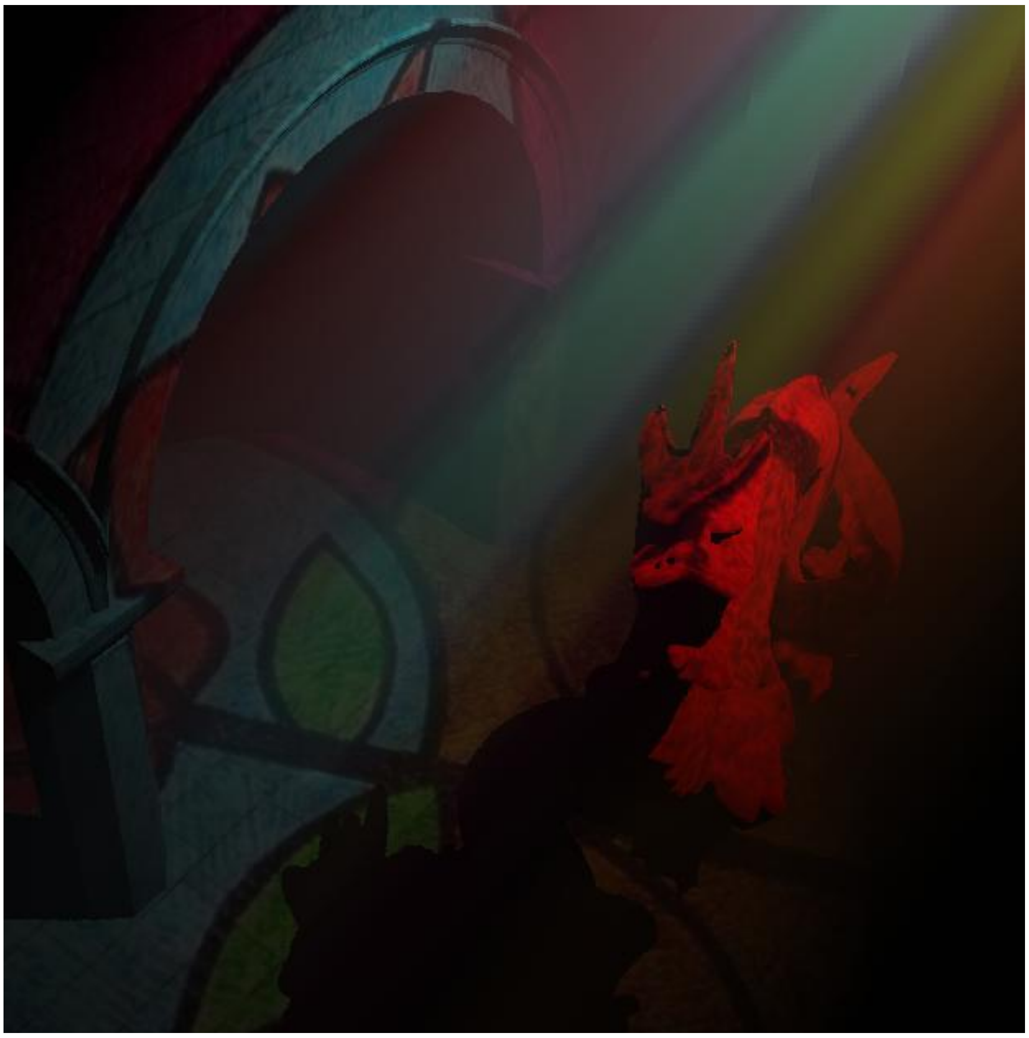


FIGURE 34 - ADDING A COLORED TEXTURE TO THE SPOT LIGHT

The scattering equation includes a phase function. This phase function describes how light is scattered through the media, and three functions have been implemented in this thesis: Isotropic scattering, the Henyey-Greenstein model, and the approximation to the Henyey-Greenstein function developed by Schlick.

The isotropic scattering function is simply:

$$p = \frac{1}{4\pi}$$

Which means that light is scattered evenly in all directions. This phase function results in very subtle scattering of light.

The Henyey-Greenstein model is defines as:

$$p = \frac{1 - g^2}{4\pi \cdot (1 + g^2 - 2g \cdot \cos(\theta))^{1.5}}$$

Where θ is the angle between a vector from the eye to the light, and a vector from the eye to the point being calculated. g is a constant that defines how light is scattered through the medium. $g < 0$ for back scattering. $g > 0$ for forward scattering, and when $g = 0$ the phase function reduces to the isotropic scattering function.

The 1.5-exponent in the Henyey-Greenstein model is quite costly, which made Schlick develop an approximation to this function. It is defined as:

$$p = \frac{1 - k^2}{4\pi \cdot (1 + k\cos(\theta))^2}$$

This approximation uses the equation of an ellipse to estimate the Henyey-Greenstein function, and is quite cheaper. Here k is similar to g in the previous model, except with the opposite sign. Plots of the two functions are shown in Figure 35

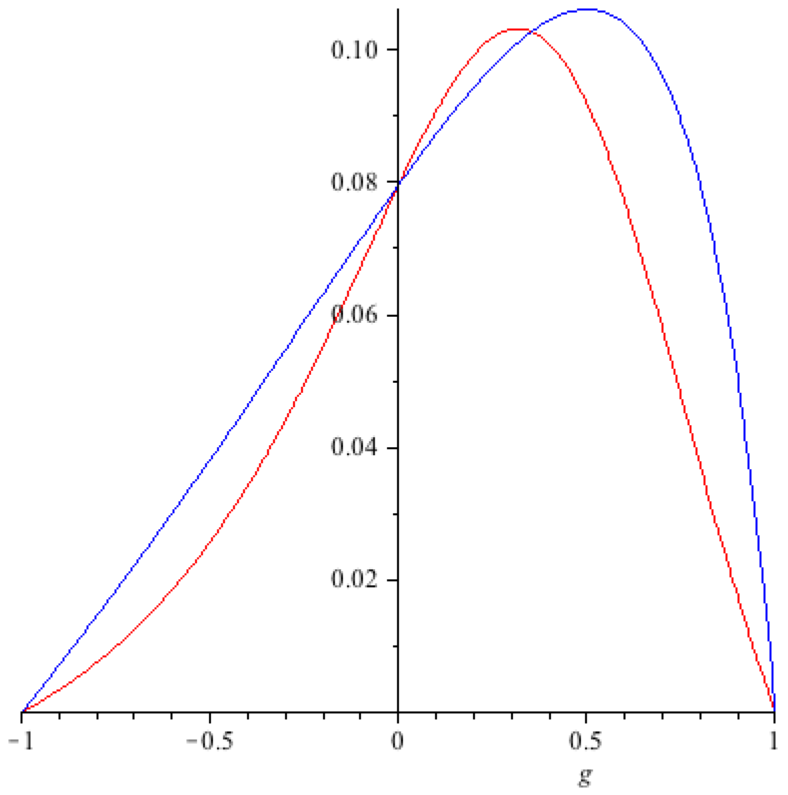


FIGURE 35 - THE HENYEY-GREENSTEIN PHASE FUNCTION (RED) AND THE SCHLICK APPROXIMATION (BLUE). SCHLICK IS PLOTTED WITH K INVERSED FOR COMPARISON

Light scattering is a quite subtle effect in reality, but when used for effect in computer games or movies, you typically want it to be very visible. Luckily, the ray marching approach is very flexible. I have implemented a simple 'artist'-factor which is multiplied on to each scattering calculation. This is obviously not physically correct in any way, but can enhance the mood you are trying to create in the scene. It can also be used to emulate the more expensive phase functions while simply using the isotropic, as shown in Figure 36

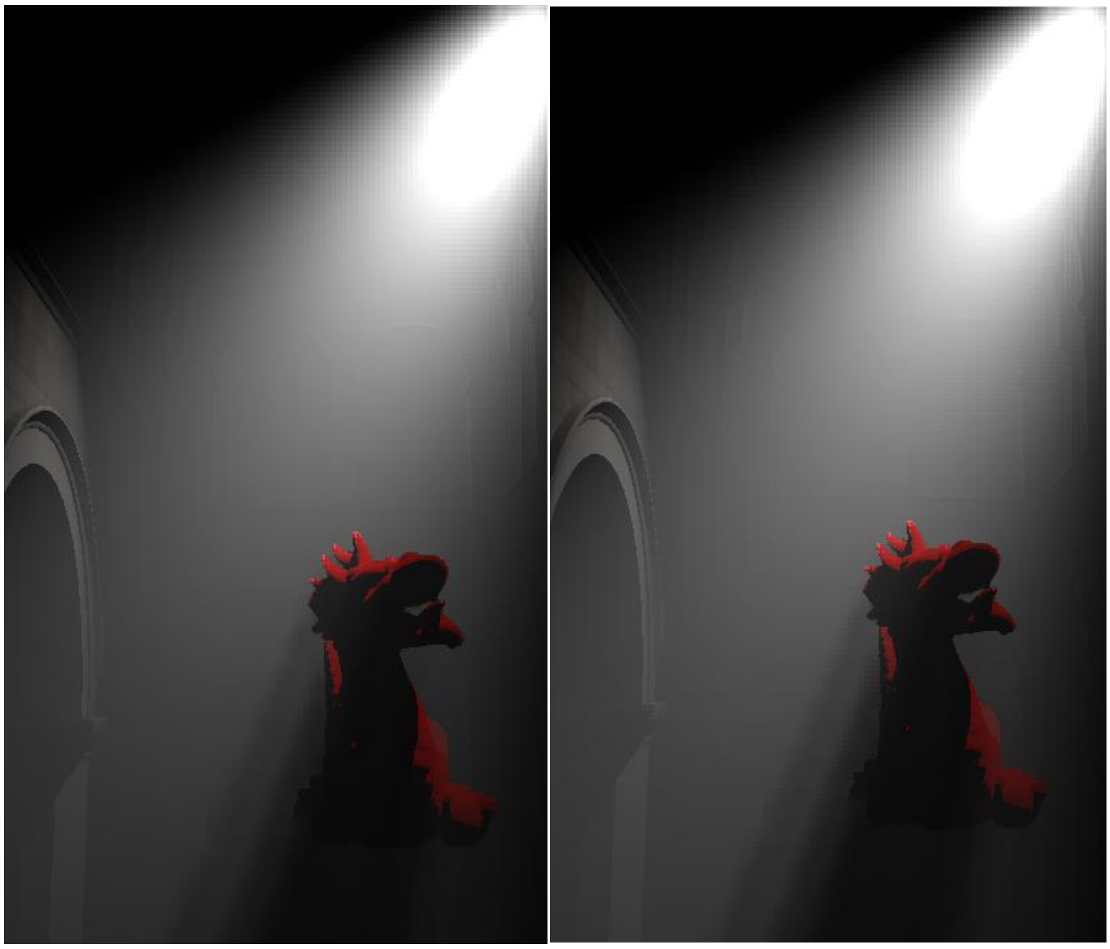


FIGURE 36 - THE SCHLICK PHASE FUNCTION (LEFT) ESTIMATED WITH THE ISOTROPIC PHASE FUNCTION, AND AN 'ARTIST'-MULTIPLIER OF 20

One additional implementation is to add support for various media. The screenshots shown so far have all been for fog, but another relevant medium is water. The difference between these two media is their scattering and absorption coefficients. I have used a three dimensional absorption coefficient to allow for varying absorption of different wavelengths of light. For fog, I have used $\sigma_a = (0.01, 0.01, 0.01)$ and for water: $\sigma_a = (0.123, 0.035, 0.006)$ giving water a blue tint. The scattering coefficient σ_s is set to 0.1 in all screenshots in this section, but can be varied as will be shown in section 5.1.

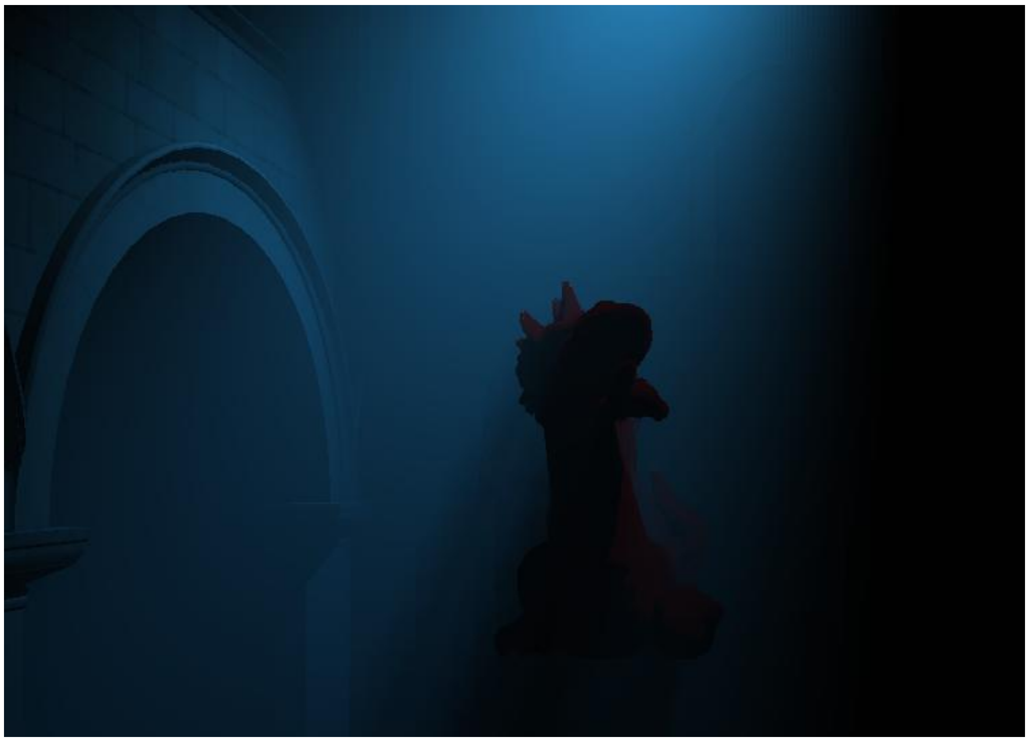


FIGURE 37 - UNDERWATER RENDERING

4.2.1 SCREENSPACE APPROACH

While the ray-marching solution works very well for closed spaces, it becomes very expensive when moving to wide open areas. Here, God Rays are quite limited, but can add a lot towards setting the mood in a scene. This has led to the development of the screenspace approach from [Mitchell 2007] which gives very nice looking results while being very cheap. It is a fast approximation, but physically quite incorrect.

Implemented using the atmospheric scattering code found at [Web - Urbano Álvarez] which is based on the work presented by CryTek in [Wenzel 2006] where the model from [Nishita et al. 1993] is solved in a two-pass fashion where the scattering texture is updated using the graphics card, and the second pass uses the scattering texture as a look-up-table to render the atmospheric scattering.

My implementation works as a fullscreen pass. I march along the vector from the current pixel to the position of the sun in screenspace-coordinates. At every point the scene depth map is sampled first. If there is nothing occluding the sky, that is if the scene depth equals 1.0, the sky texture is sampled. This is done several times, stepping along the vector and adding up the un-occluded samples. These samples are attenuated based on the distance from the sun. The resulting God ray-image is then superimposed over the regular scene rendering as shown in Figure 38.

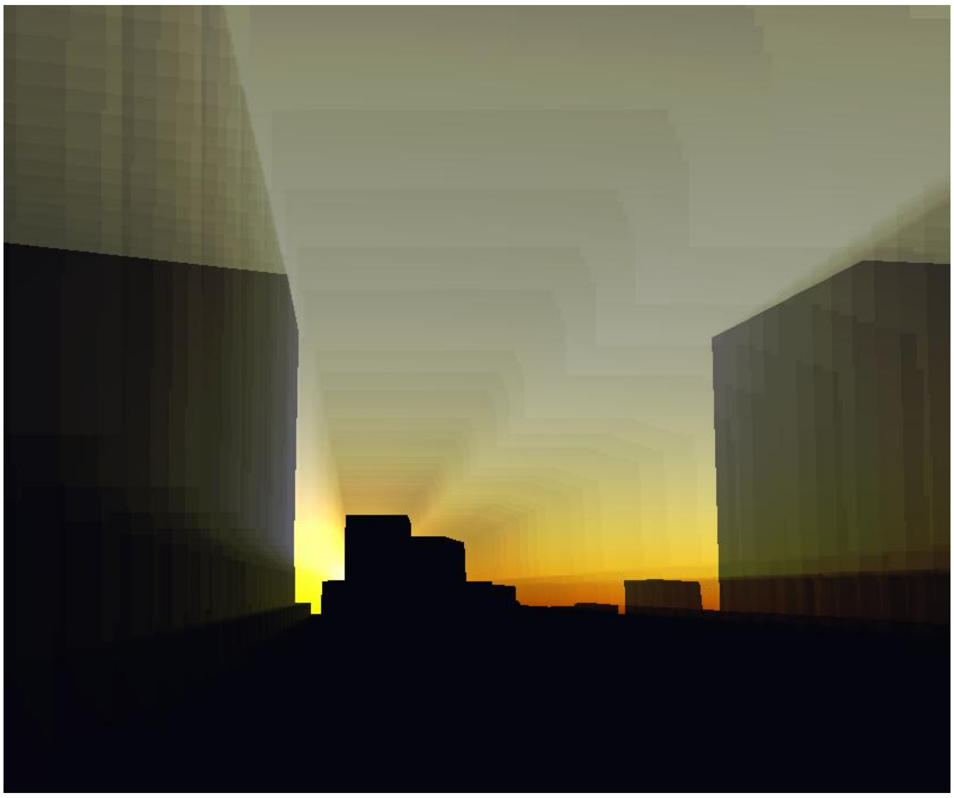


FIGURE 38 - RAY-MARCHING IN SCREENSPACE. THE EFFECT HAS BEEN ENHANCED TO MAKE FOR A BETTER SCREENSHOT

It is worth noting that a cosine factor is multiplied on to avoid God rays when looking directly away from the sun. In this case, the sun will have the same x and y coordinates in screenspace. This cosine factor can also be altered to make the God rays more visible when looking directly at the sun, which seems more realistic.

4.3 Subsurface scattering

For general subsurface scattering, I have implemented the method presented in [Ki 2009] as it works well with the deferred approach to lighting. An extended shadow map is rendered from the point of view of the light source, containing (in addition to scene depth) the surface normal; the irradiance entering the model (that is the light intensity attenuated by distance and by the Fresnel function) and finally the world-space position. A material ID can also be included to allow for scenes with multiple materials, but is not relevant for my sample program.

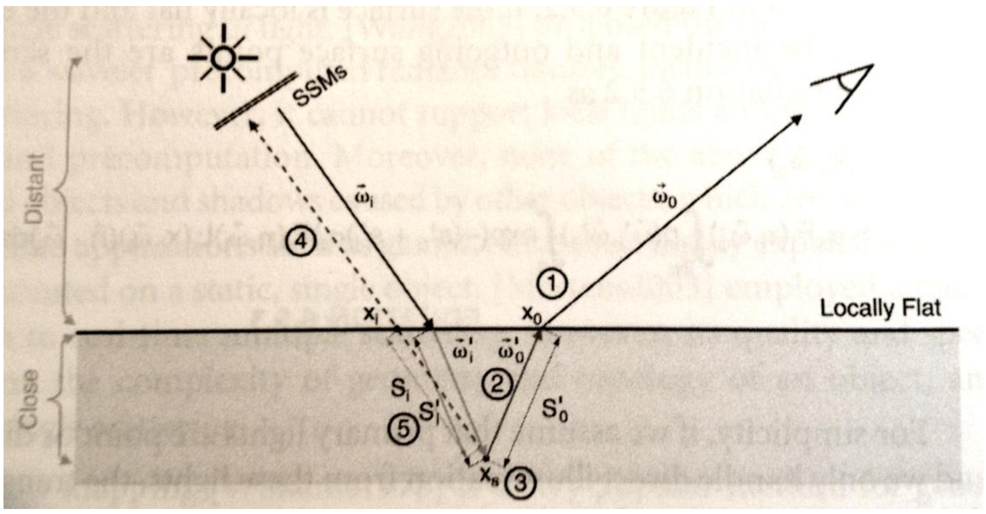
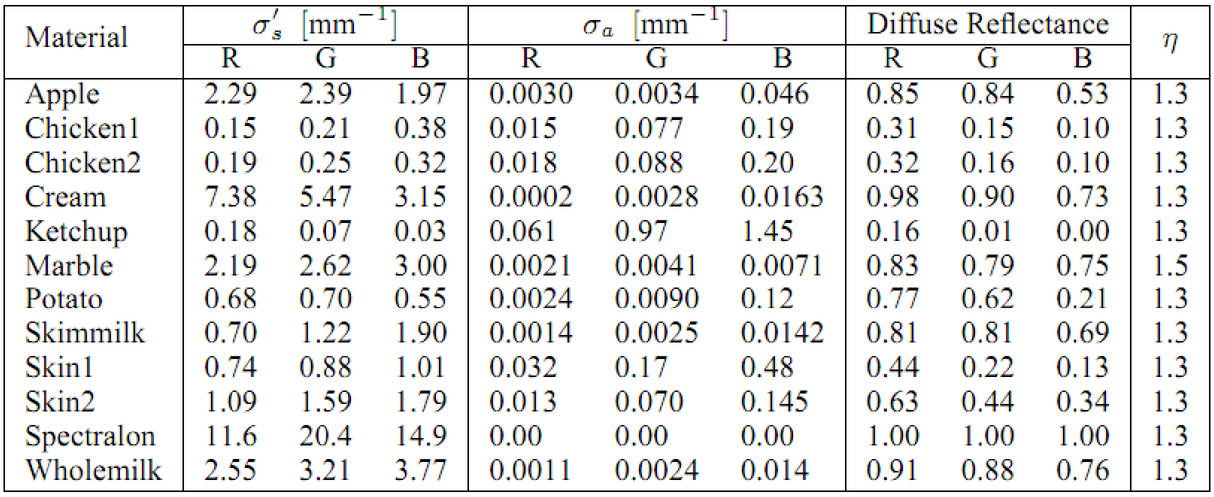


FIGURE 39 - GENERAL ALGORITHM FROM [KI 2009]

The algorithm shown in Figure 39 works by ray-marching through the model along the outgoing path s'_o and computing the subsurface intersection x_s. This point is then transformed to light space coordinates to find the surface point x_i. The information stored in the Subsurface Scatter Map can then be used to calculate s_i giving a good approximation to s'_i.

The method uses scattering coefficients found in the table in Figure 40 and examples can be seen in Figure 41 and Figure 42 for skim milk and ketchup respectively



Material	σ'_s [mm^{-1}]			σ_a [mm^{-1}]			Diffuse Reflectance			η
	R	G	B	R	G	B	R	G	B	
Apple	2.29	2.39	1.97	0.0030	0.0034	0.046	0.85	0.84	0.53	1.3
Chicken1	0.15	0.21	0.38	0.015	0.077	0.19	0.31	0.15	0.10	1.3
Chicken2	0.19	0.25	0.32	0.018	0.088	0.20	0.32	0.16	0.10	1.3
Cream	7.38	5.47	3.15	0.0002	0.0028	0.0163	0.98	0.90	0.73	1.3
Ketchup	0.18	0.07	0.03	0.061	0.97	1.45	0.16	0.01	0.00	1.3
Marble	2.19	2.62	3.00	0.0021	0.0041	0.0071	0.83	0.79	0.75	1.5
Potato	0.68	0.70	0.55	0.0024	0.0090	0.12	0.77	0.62	0.21	1.3
Skimmilk	0.70	1.22	1.90	0.0014	0.0025	0.0142	0.81	0.81	0.69	1.3
Skin1	0.74	0.88	1.01	0.032	0.17	0.48	0.44	0.22	0.13	1.3
Skin2	1.09	1.59	1.79	0.013	0.070	0.145	0.63	0.44	0.34	1.3
Spectralon	11.6	20.4	14.9	0.00	0.00	0.00	1.00	1.00	1.00	1.3
Wholemilk	2.55	3.21	3.77	0.0011	0.0024	0.014	0.91	0.88	0.76	1.3

FIGURE 40 - TABLE OF MATERIAL PROPERTIES FROM [JENSEN ET AL. 2001]

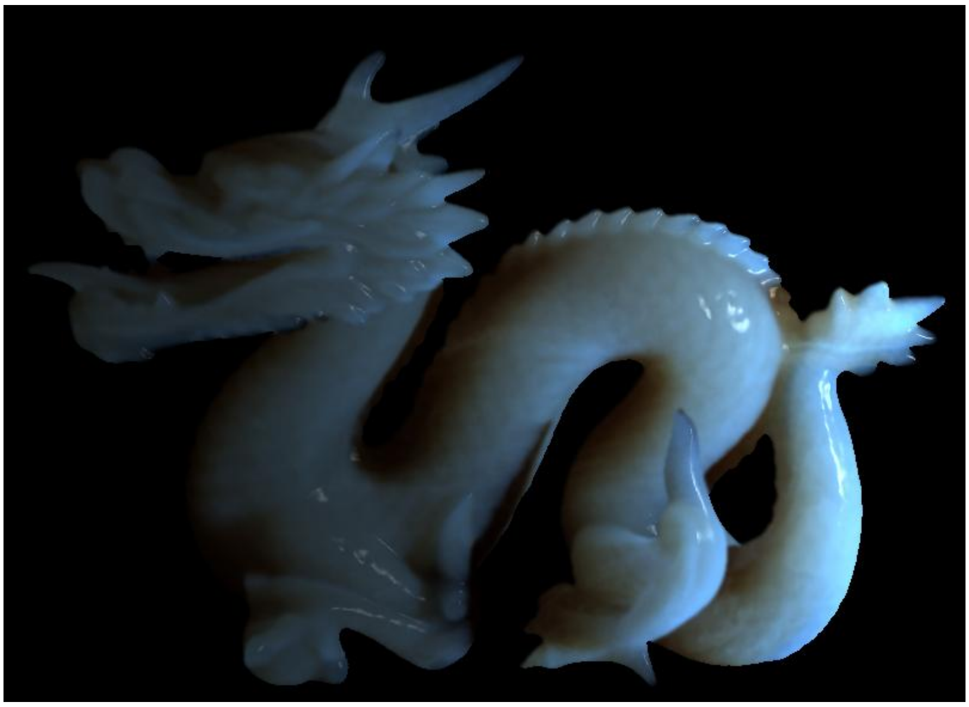


FIGURE 41 - SUBSURFACE SCATTERING USING MATERIAL PROPERTIES FOR SKIM MILK

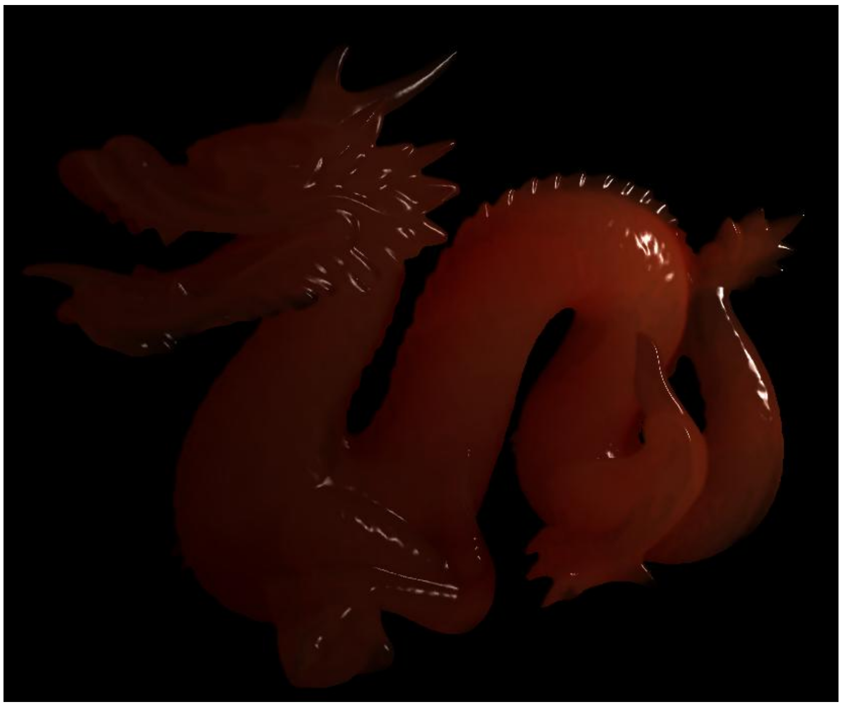


FIGURE 42 - SUBSURFACE SCATTERING USING MATERIAL PROPERTIES FOR KETCHUP. NOTE THAT THE LIGHT INTENSITY HAS BEEN INCREASED TO BRIGHTEN UP THE SCREENSHOT

I have implemented a simple Gaussian blur of the scattered part of the lighting, as it exhibits noise artifacts from under-sampling. Subsurface scattering also has an inherently smooth look, which is enhanced by the blurring. The effect of blurring is shown in Figure 43.

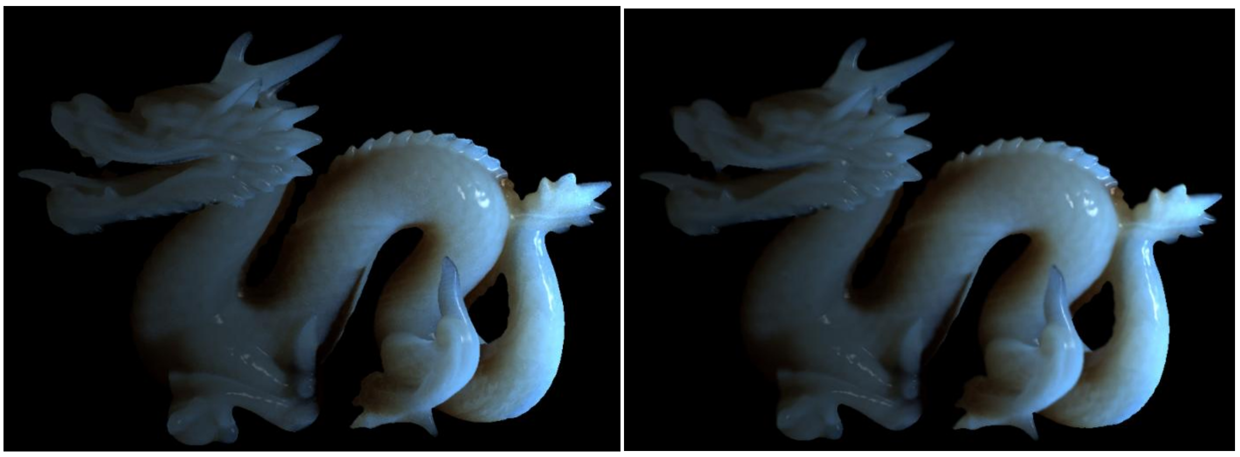


FIGURE 43 - BLURRING THE SUBSURFACE SCATTERING CONTRIBUTION (RIGHT) REMOVES NOISE FROM UNDER-SAMPLING (LEFT)

4.3.1 Skin rendering

For my skin rendering solution, I have implemented the algorithm from [d'Eon and Luebke 2007] as it is widely acknowledged as being the very best, when it comes to image quality.

When modeling multi-layered materials like skin, the notion of light entering and exiting the same spot on a surface, is not exact enough. Light enters the material, scatters and exits a different place. This scattering is different for different wavelengths of light, and is illustrated in Figure 44 for human skin.

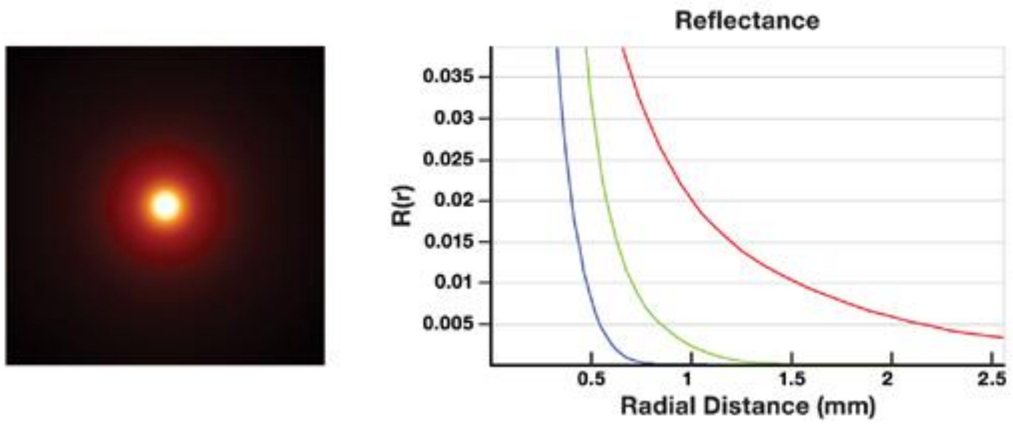
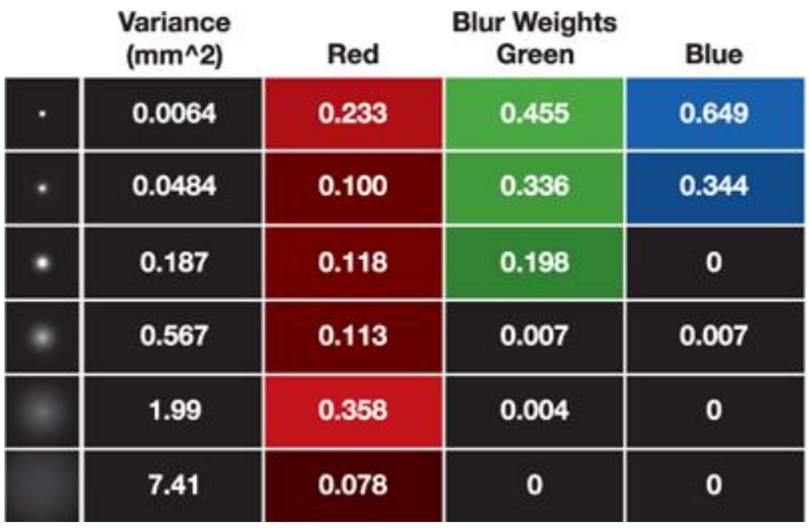


FIGURE 44 - DIFFUSION PROFILE FOR SKIN. IMAGE FROM [D'EON AND LUEBKE 2007]

[d'Eon and Luebke 2007] found that they could almost match the diffusion profiles with the common Gaussian function e^{-r^2}, which is very simple to calculate in computer graphics. The Gaussian function doesn't match it precisely, but when expanded to a sum of Gaussians, the result is very close. They have arrived at the following values, when trying to match the three-layer skin model from [Donner and Jensen 2005]:



	Variance (mm^2)	Blur Weights Red	Green	Blue
●	0.0064	0.233	0.455	0.649
●	0.0484	0.100	0.336	0.344
●	0.187	0.118	0.198	0
●	0.567	0.113	0.007	0.007
●	1.99	0.358	0.004	0
●	7.41	0.078	0	0

FIGURE 45 - GAUSSIAN BLUR KERNELS AND WEIGHTS (FROM [D'EON AND LUEBKE 2007])

Gaussian blur functions are used for many things in real-time graphics because of their simplicity and speed. The function is separable in x and y allowing for even faster calculation with a two-pass approach (see Figure 46). This is done by first rendering the horizontal blur to an intermediate render target. And then blurring this intermediate render target vertically, outputting the final result.

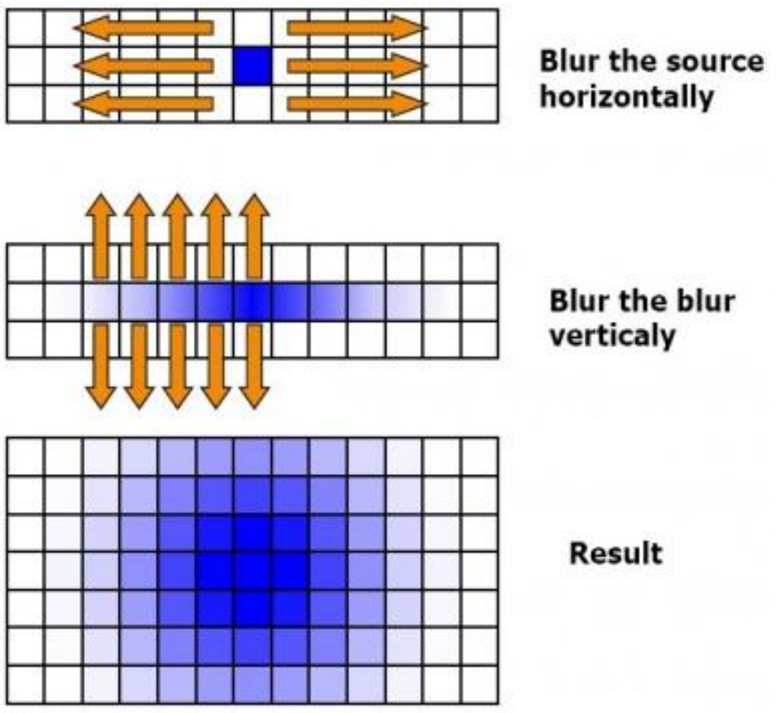


FIGURE 46 - ILLUSTRATION OF SEPARABLE BLURRING. IMAGE FROM [ENGEL 2003]

In most real-time computer graphics, the Phong reflectance model is the BRDF of choice. While it is fast, more complex materials such as skin require a more advanced BRDF. For skin rendering, the physically based Kelemen/Szirmay-Kalos BRDF has shown to give very pleasing results. This is the shader code used to calculate the BRDF based on surface normal, light vector, eye vector, roughness and specular power. The Fresnel reflectance calculation is done by using the Schlick approximation.

```
float KS_specular(float3 N, // Surface normal
                  float3 L, // point-to-light vector
                  float3 V, // Point-to-eye vector
                  float m, // roughness
                  float rho_s) // specular brightness
{
    float result = 0.0;
    float ndotl = dot( N, L );
    if( ndotl > 0.0 )
    {
        float3 h = L + V; // Unnormalized half-way vector
        float3 H = normalize( h );
        float ndoth = dot( N, H );

        float alpha = acos( ndoth );
        float ta = tan( alpha );
        float PH = 1.0/(m*m*pow(ndoth,4.0))*exp(-(ta*ta)/(m*m));

        float F = FresnelReflectance( H, V, 0.028 );
        float frSpec = max( (PH * F) / dot( h, h ), 0 );
        result = ndotl * rho_s * frSpec;
    }
    return result;
}
```

This BRDF is more complex to calculate, but the results are much better than what can be achieved with the standard Phong BRDF:

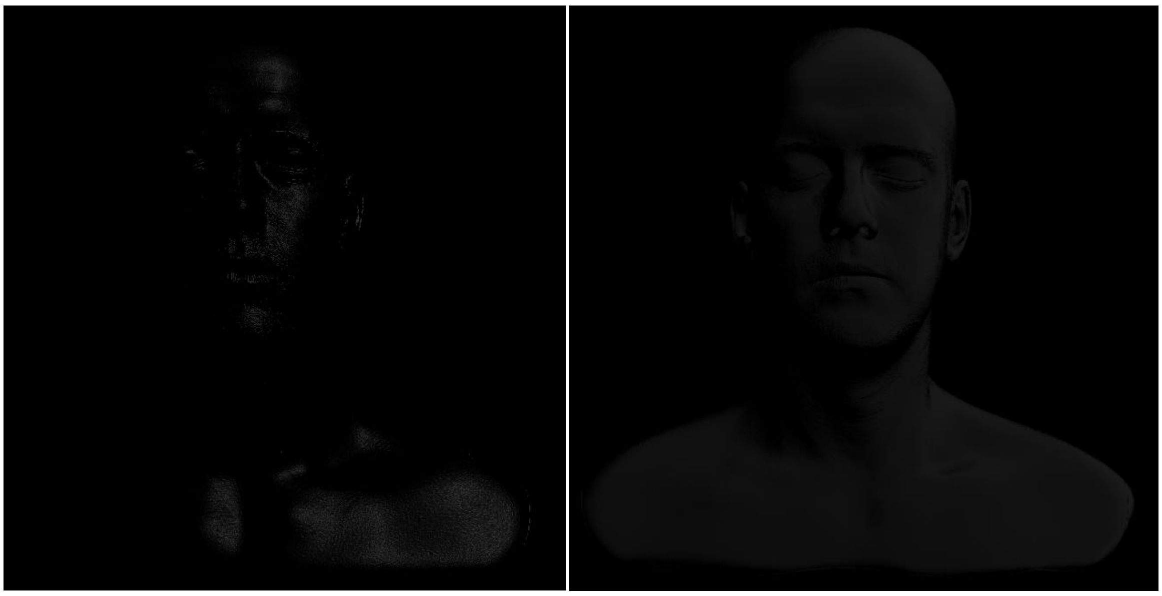
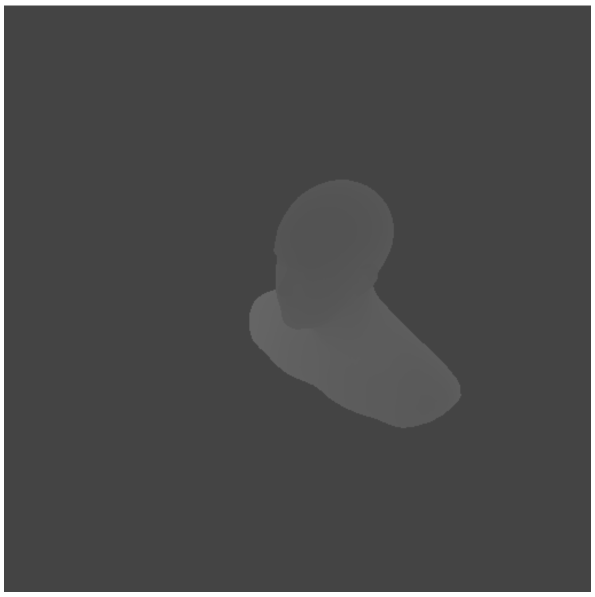


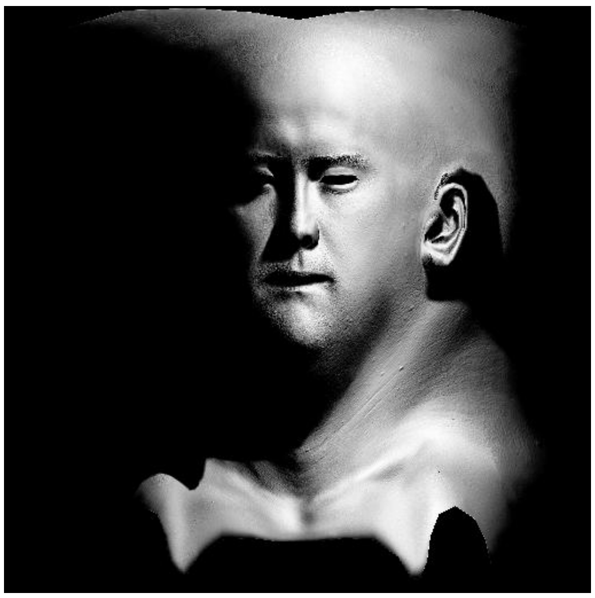
FIGURE 47 - DIFFERENCE BETWEEN SPECULAR COMPONENT OF PHONG (LEFT) AND KELEMEN/SZIRMAY-KALOS (RIGHT)

The steps of the algorithm are as follows:

The first step is to render a depth map as seen from the light source for shadow mapping



The second step is to render a light map as seen from the camera, but drawn to a render target using the texture coordinates of the model.



Steps 3 through 7 are blurs of the light map using a Gaussian blur kernel with increasing variance. Each result is stored in a texture

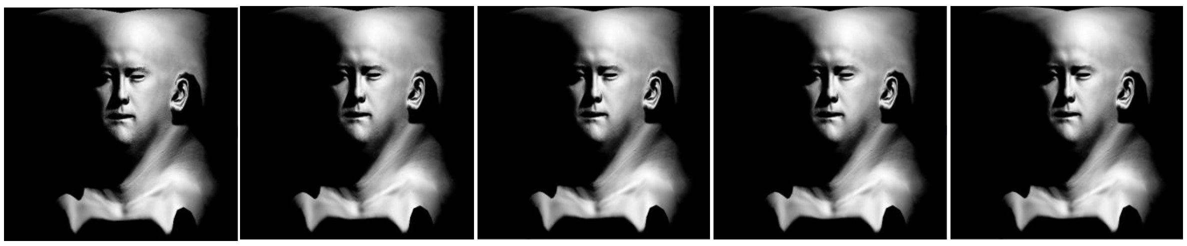


FIGURE 48 - THE BLURRED LIGHT MAPS RESULTING FROM GAUSSIAN BLURRING WITH THE WEIGHTS FROM FIGURE 45. VARIANCE INCREASING FROM LEFT TO RIGHT

Step 8 is combining these blurred light maps, where each texture is weighted separately in the red, green and blue channels as illustrated in Figure 45

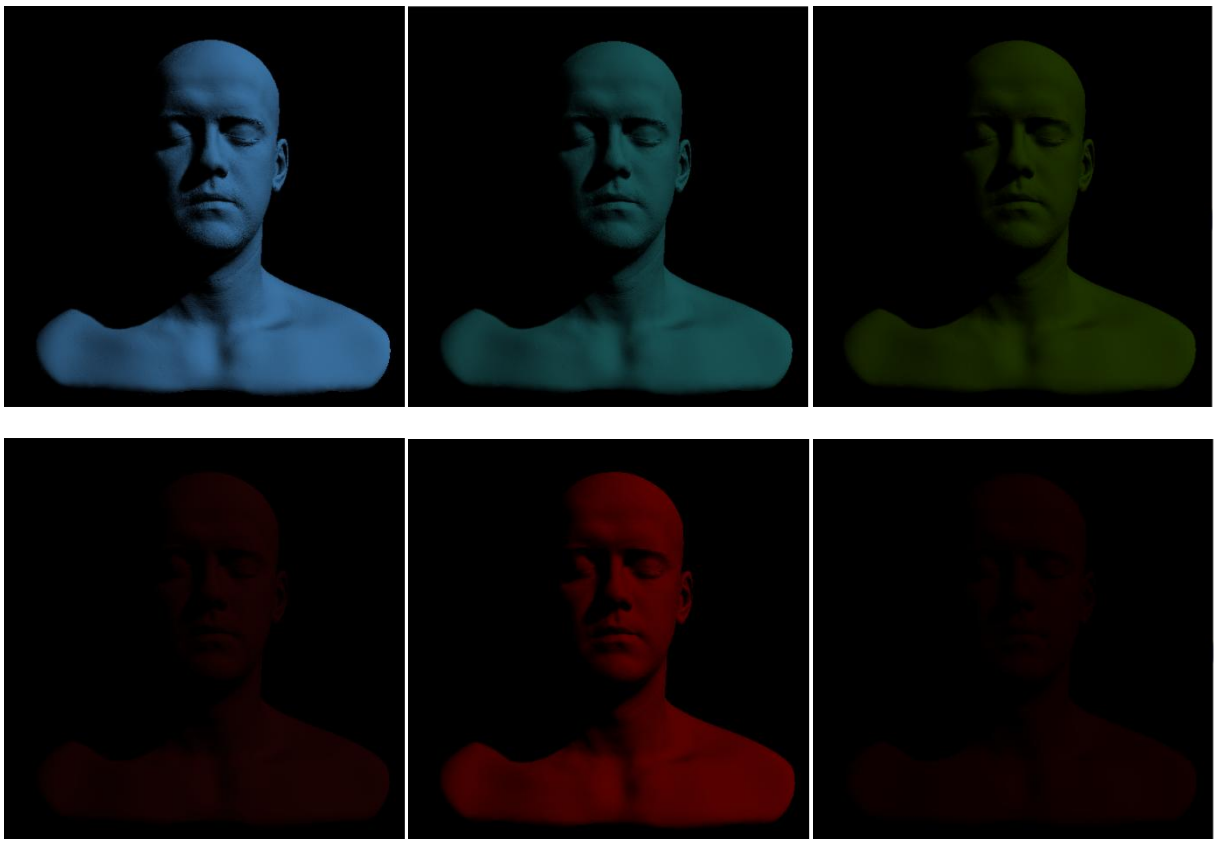


FIGURE 49 - WEIGHTED BLURRED LIGHTMAPS RENDERED ON TO THE MODEL. FROM LEFT TO RIGHT, TOP TO BOTTOM, WEIGHTED ACCORDING TO THE VALUES IN FIGURE 45

And finally step 9 is combining the blurred light maps and using them to illuminate the diffuse texture. Finally, specular highlights are added.

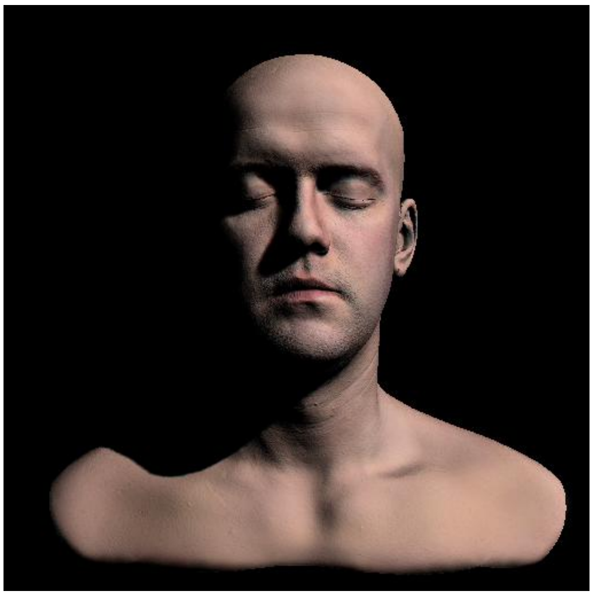


FIGURE 50 - FINAL RENDERING OF 5-PASS BLURRED TEXTURE SPACE DIFFUSION

As can be seen in Figure 50, this method gives very pleasing results, but the price is very high. The 5-pass Gaussian blurring is very computationally heavy, and the technique is still out of reach for current graphics hardware, if the scene is more complex.

With the major hurdle being the 5-pass blur, the technique presented in [Hable et al. 2009] was the next algorithm implemented. Here the light map is rendered to texture space as in the previous method, but instead of doing the multiple blur passes, the blurring is done when rendering the final pass.

The blurring is done by sampling the light map 12 times around the point in question. These 12 samples are jittered around the current texel, and weighted accordingly. The texture coordinate offsets for all the samples are:

```
float2 blurJitteredSamples[13] =
{
        { 0.000000, 0.000000 },
        { 1.633992, 0.036795 },
        { 0.177801, 1.717593 },
        { -0.194906, 0.091094 },
        { -0.239737, -0.220217 },
        { -0.003530, -0.118219 },
        { 1.320107, -0.181542 },
        { 5.970690, 0.253378 },
        { -1.089250, 4.958349 },
        { -4.015465, 4.156699 },
        { -4.063099, -4.110150 },
        { -0.638605, -6.297663 },
        { 2.542348, -3.245901 },
};
```

And the weighting of each sample with individual weights for each color are:

```
float3 blurJitteredWeights[13] =
{
        { 0.220441, 0.437000, 0.635000 },
        { 0.076356, 0.064487, 0.039097 },
        { 0.116515, 0.103222, 0.064912 },
        { 0.064844, 0.086388, 0.062272 },
        { 0.131798, 0.151695, 0.103676 },
        { 0.025690, 0.042728, 0.033003 },
        { 0.048593, 0.064740, 0.046131 },
        { 0.048092, 0.003042, 0.000400 },
        { 0.048845, 0.005406, 0.001222 },
        { 0.051322, 0.006034, 0.001420 },
        { 0.061428, 0.009152, 0.002511 },
        { 0.030936, 0.002868, 0.000652 },
        { 0.073580, 0.023239, 0.009703 },
};
```

This method is a lot faster than the full 5-pass rendering, and gives very comparable results. They are not as correct when studied very closely, but in real-time applications where the humans are merely part of a larger scene, you will be hard pressed to notice the difference.

Looking to screen-space for further speed improvements, I chose a simple filter based roughly on the one found in [d'Eon and Luebke 2007], but greatly simplified. The purpose of this was mainly to illustrate that even a very simple weighted smoothing of the diffuse lighting adds a lot toward more natural looking skin. The technique works by rendering the diffuse lighting contribution and storing it in a screen-space buffer as in normal light pre-pass rendering. This light buffer is then passed through a two-pass separable Gaussian blur filter, using the variance found in Figure 51. Then when combining the final image, the un-blurred light map (blurred with variance 0) and the blurred light map are sampled using the weights shown in Figure 51, and the specular term is calculated and added.

Variance (mm^2)	Red weight	Green weight	Blue weight
0	0.233	0.455	0.694
6	0.767	0.545	0.351

FIGURE 51 - SIMPLE GAUSSIAN WEIGHTS FOR SCREEN-SPACE DIFFUSION

While this method is obviously not physically correct, it goes to show that even a rough estimate makes skin rendering much more pleasing to the human eye. A comparison to plain Lambertian diffuse shading can be seen in the discussion of results in section 5.3.

The head model in this part of the thesis is from [Web - Infinite Realities]

5 RESULTS

This chapter will show the results obtained in this thesis. All results are completely interactive and properties can be changed on the fly.

5.1 PARTICIPATING MEDIA

This effect is still a bit expensive for current video games, but it is getting closer. With the ray marching approach, there are so many different optimizations to implement, that it is increasingly feasible to include in video games. Many of these optimizations work especially well under certain circumstances and others well in different situations, making this a flexible and adaptable approach.

Images of the ray marching solution under various conditions:

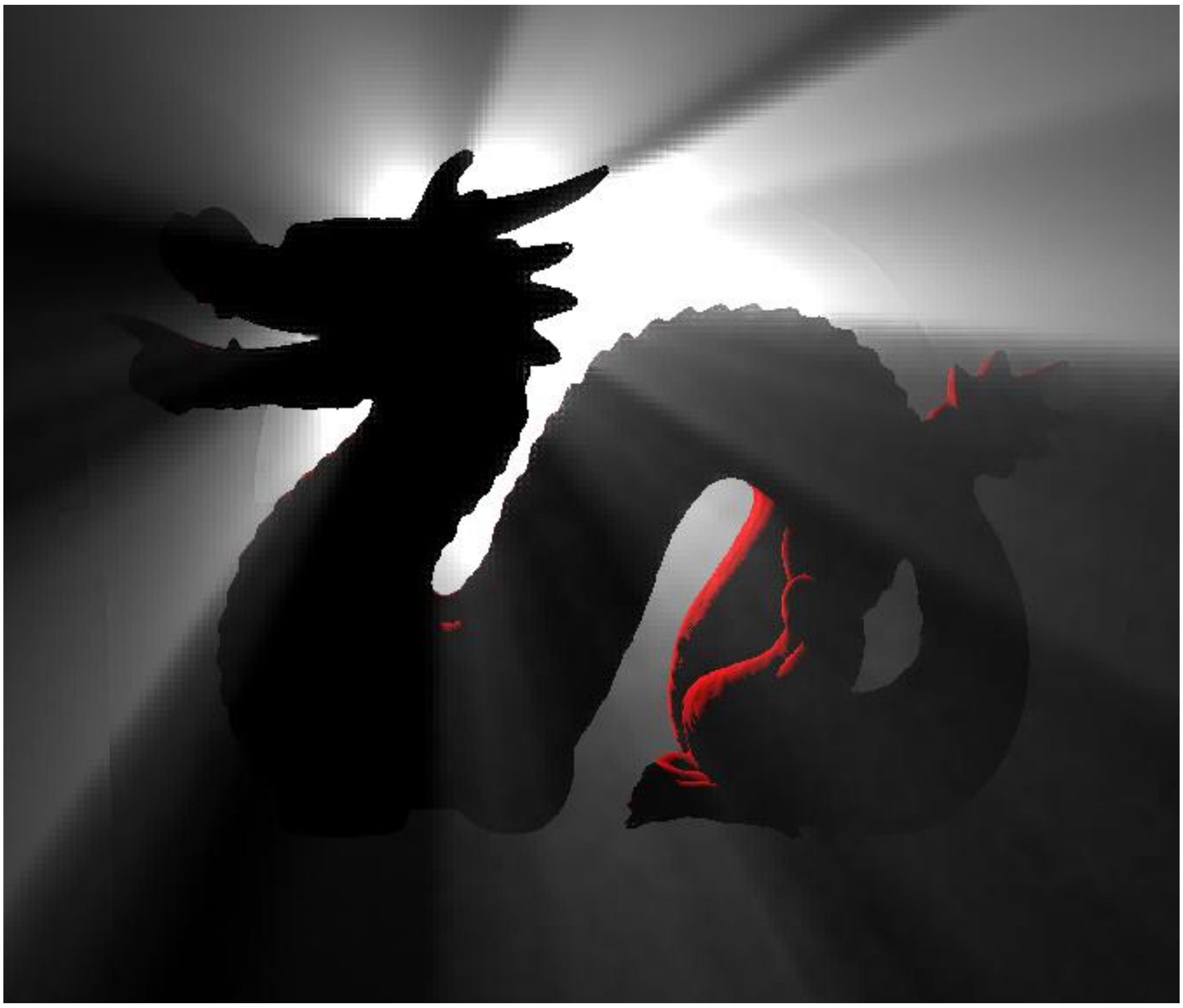


FIGURE 52 - VOLUMETRIC SHADOWS IN FOG

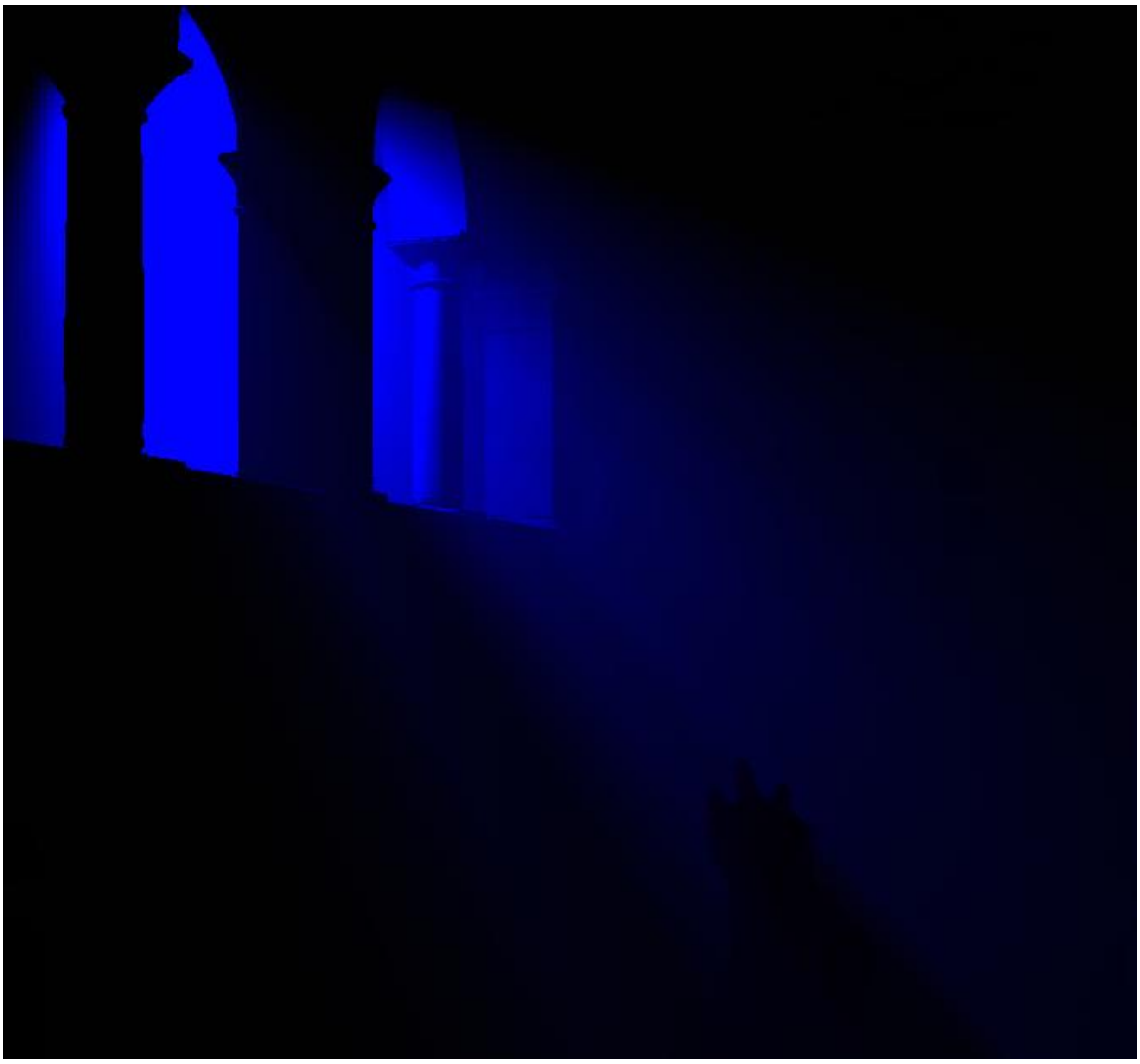


FIGURE 53 - A BLUE LIGHT SOURCE CASTING BEAMS OF LIGHT THROUGH FOG

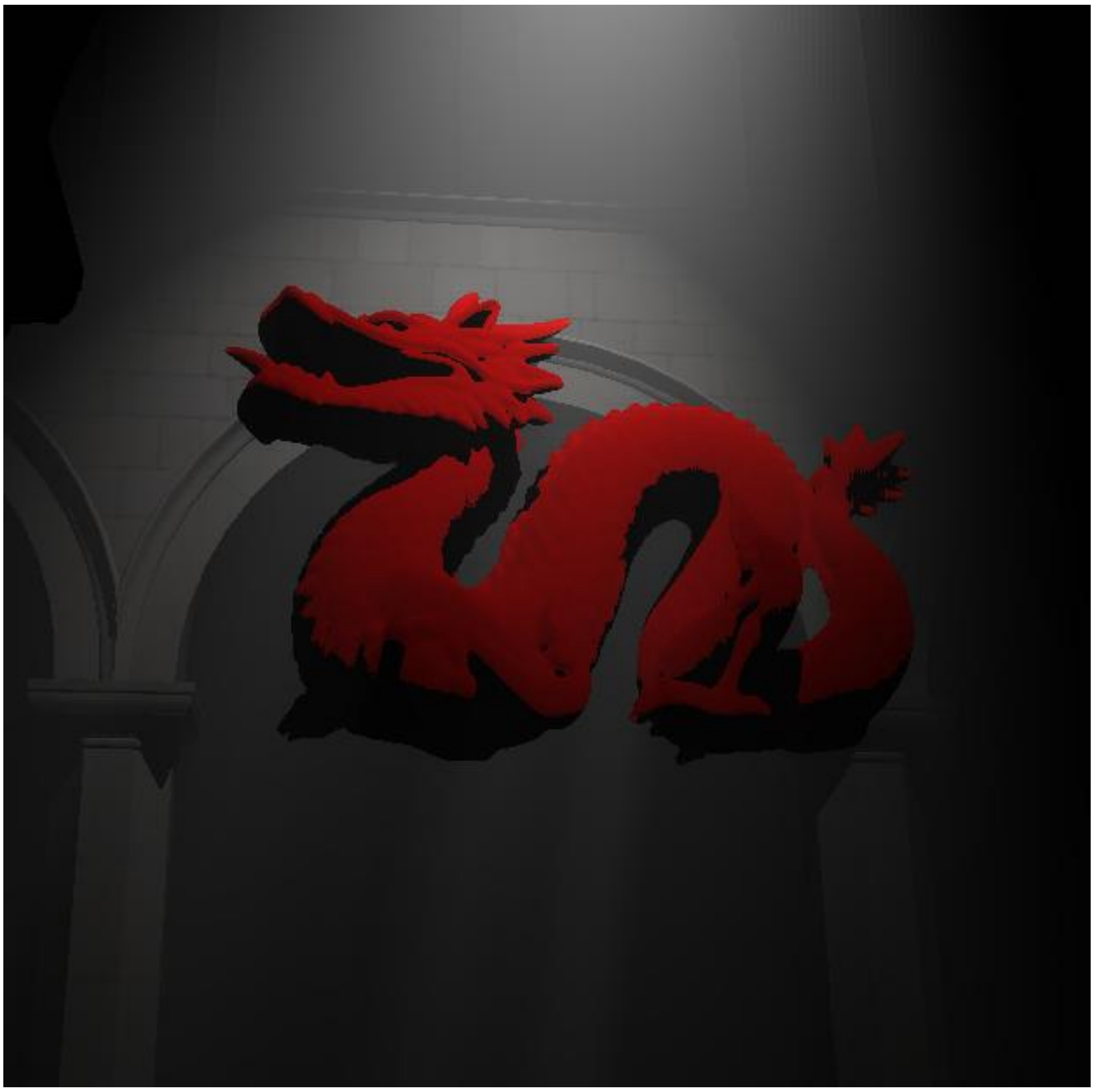


FIGURE 54 - UNDER SOME CONDITIONS, VOLUMETRIC SCATTERING IS A VERY SUBTLE EFFECT

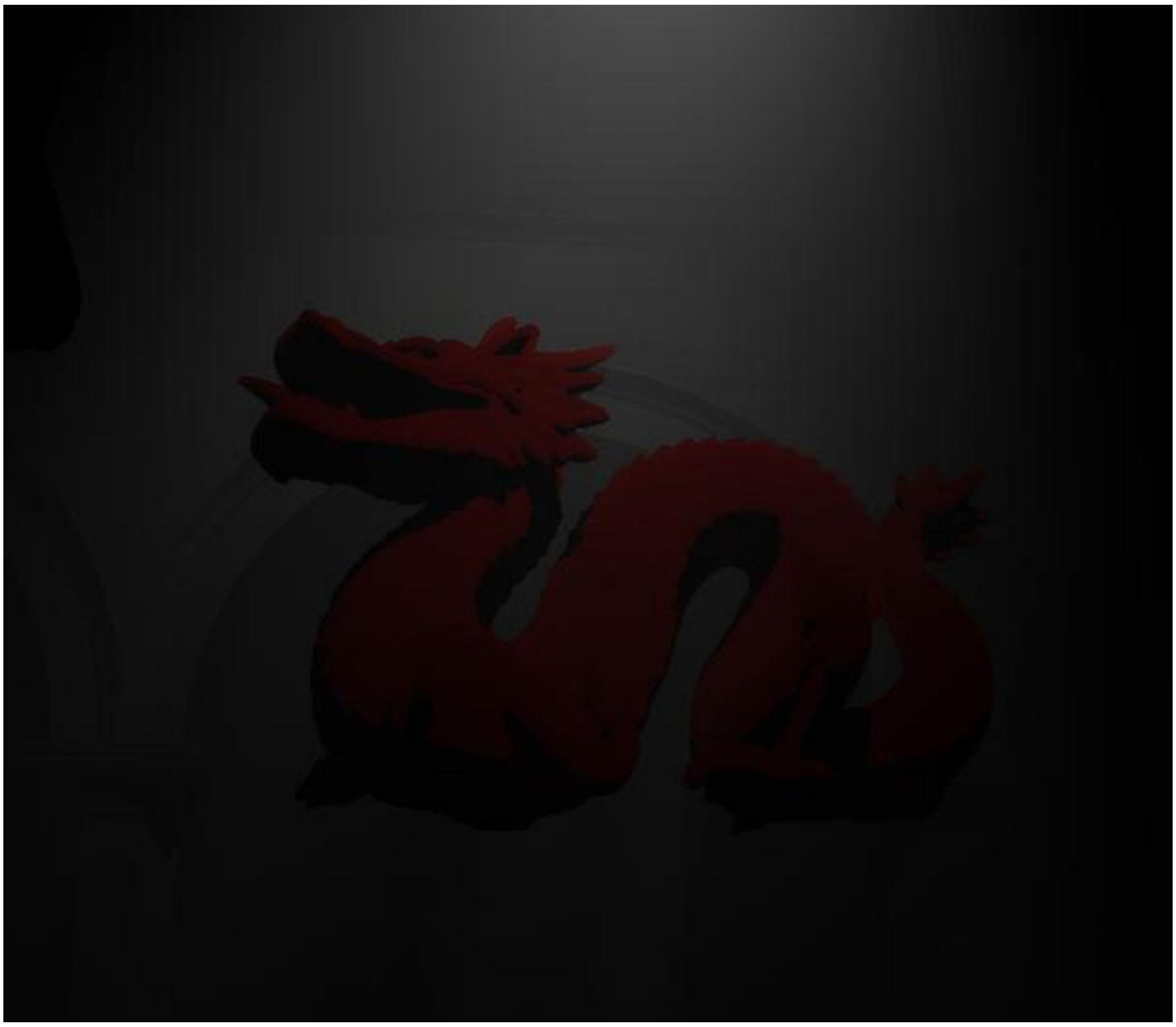


FIGURE 55 - SAME AS ABOVE, BUT WITH A SCATTERING COEFFICIENT OF 0.2

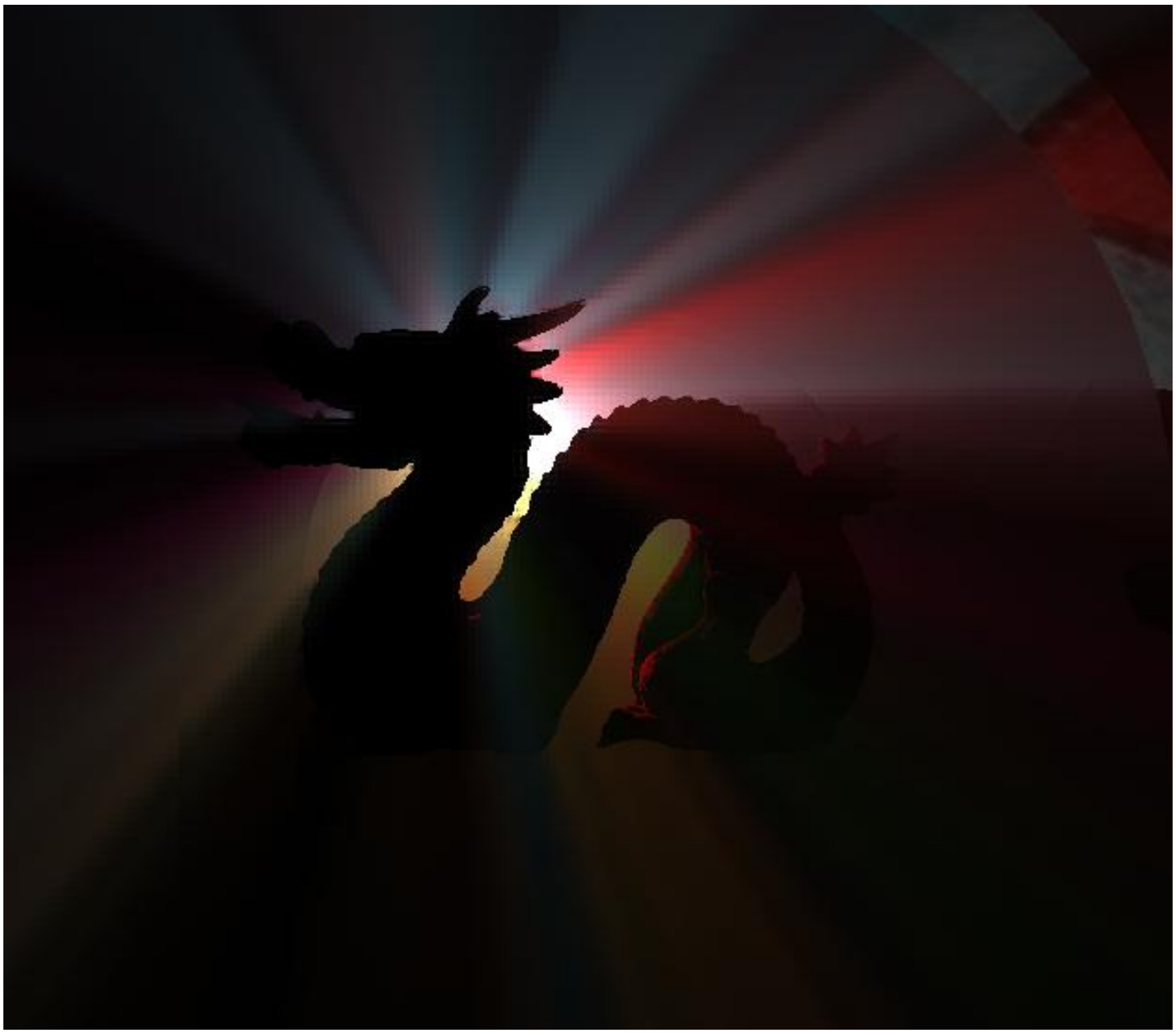


FIGURE 56 - ADDING A TEXTURED LIGHT SOURCE CAN GIVE A POWERFUL EFFECT

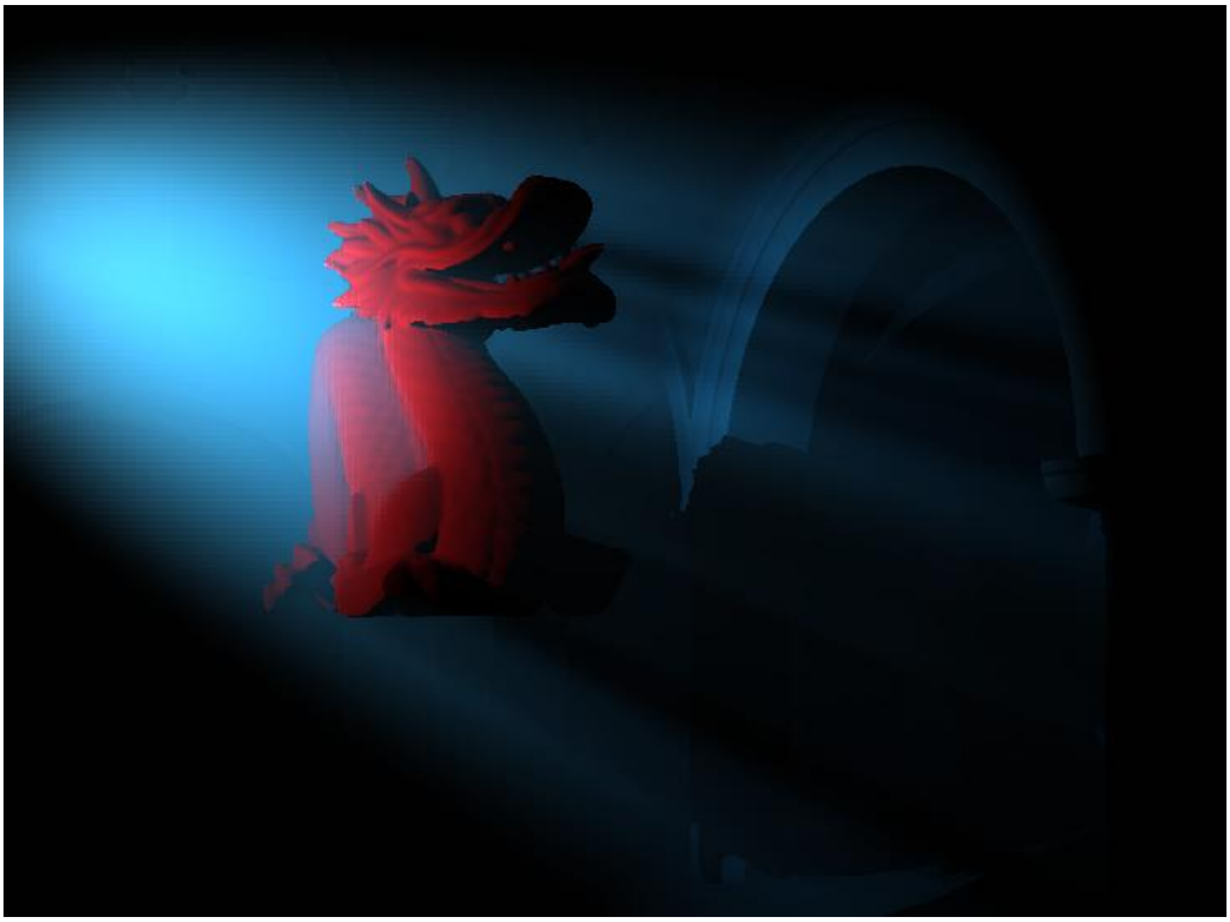


FIGURE 57 - ABSORPTION COEFFICIENT CHANGED TO SIMULATE UNDERWATER RENDERING

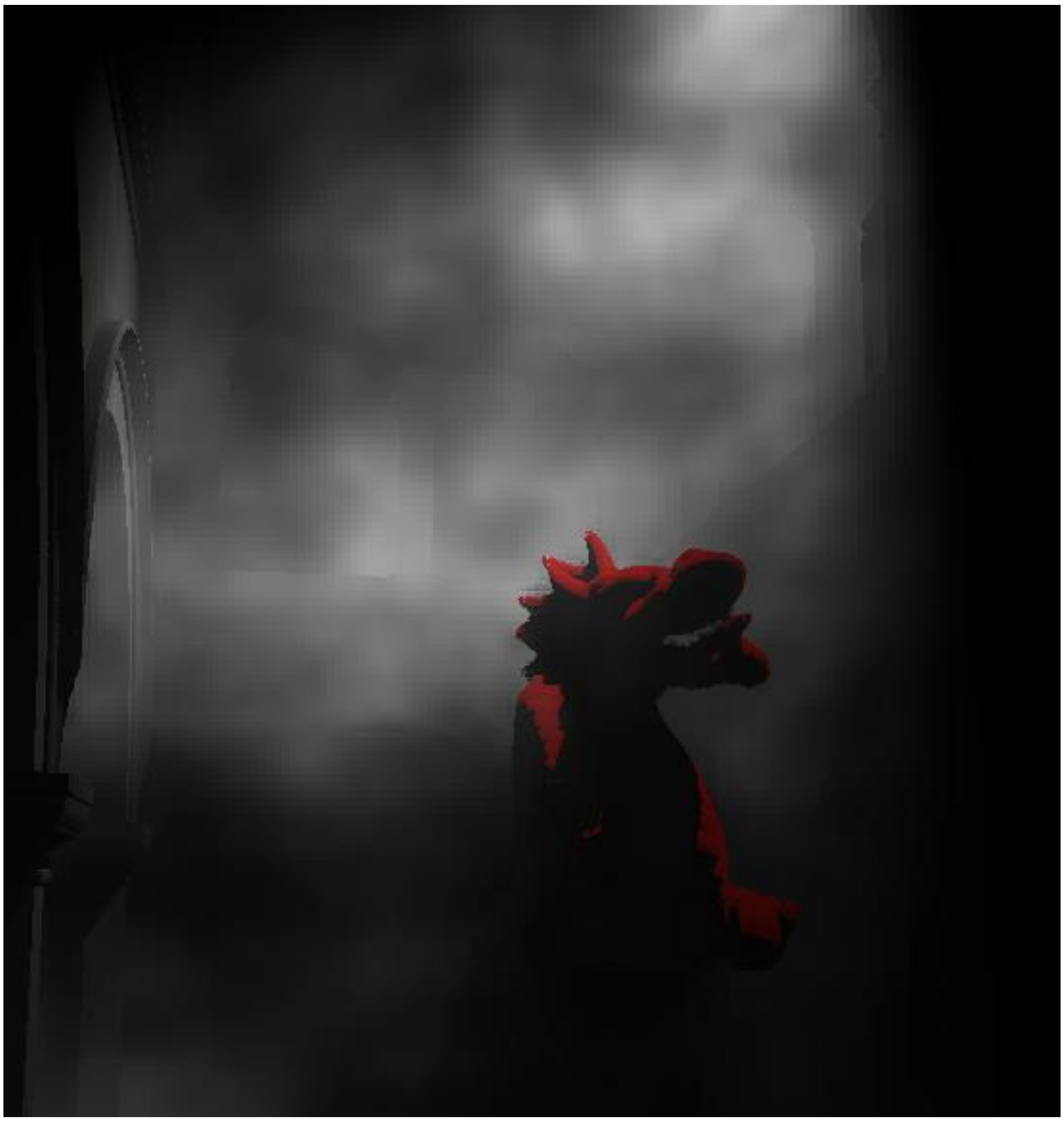


FIGURE 58 - SIMPLE PERLIN NOISE ADDED TO EMULATE HETEROGENEOUS MEDIA

Many different effects can be achieved with this quite versatile technique.

5.1.1 SCREENSPACE APPROACH

The screenspace approach to God ray rendering is quite simple and fast. This is both a strength in some cases, and a weakness in other cases.

Here are some results from my work:

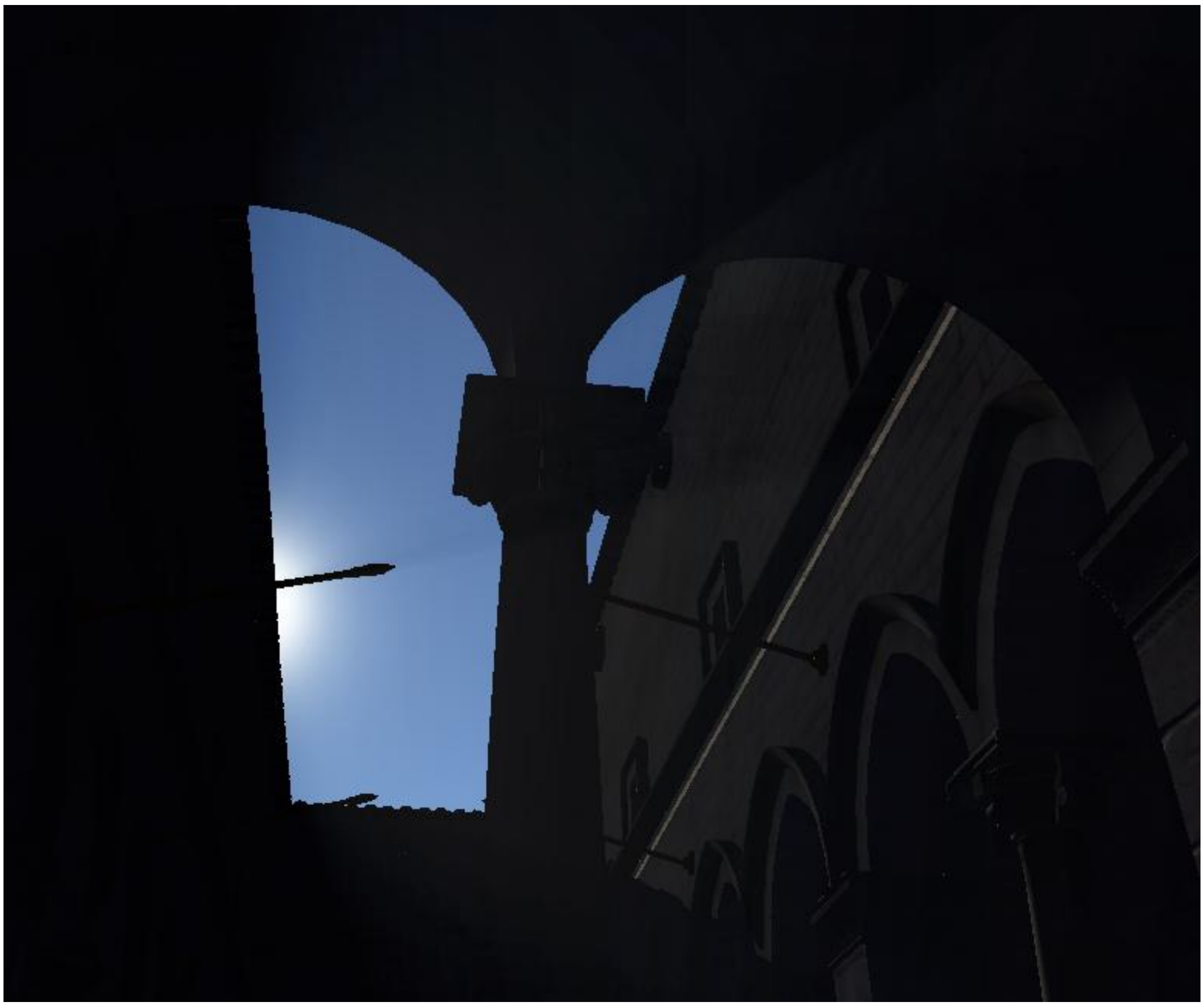


FIGURE 59 - SCREENSPACE GOD RAYS IN THE CRYTEK SPONZA SCENE

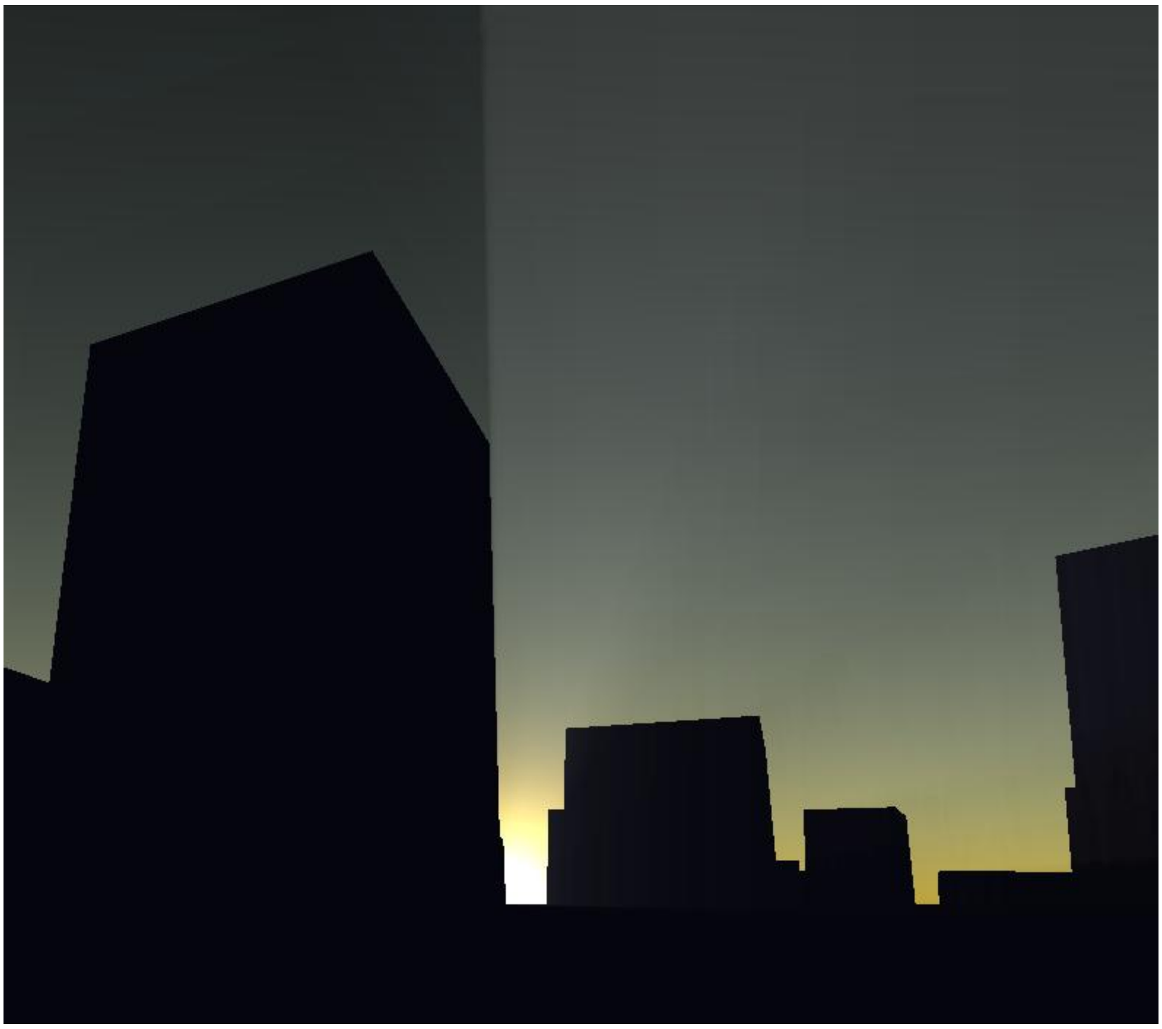


FIGURE 60 - SUNSET BEHIND SKYSCRAPERS

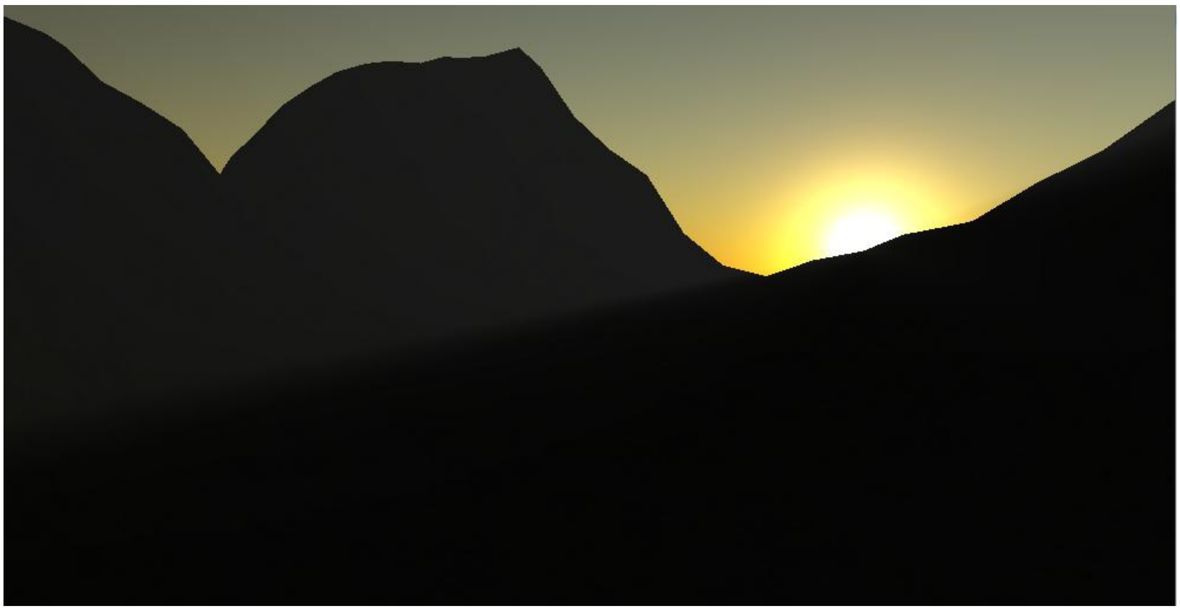


FIGURE 61 - SUNRISE BEHIND MOUNTAINS

As can be seen, adding God rays to a scene can be a powerful mood-enhancer.

While this approach works very well for these scenarios, the simplicity also has limitations. Objects right in front of the camera will have a big effect on the God rays which can look very odd. Also if used on light sources other than the sun, objects behind these light sources will also cast volumetric shadows.

5.2 Subsurface scattering

The method from [Ki 2009] has been implemented and tested for different material properties. While it can give good results, it is quite expensive and therefore slow. It also has limitations when the geometry becomes more complex. These are some results from my implementation:

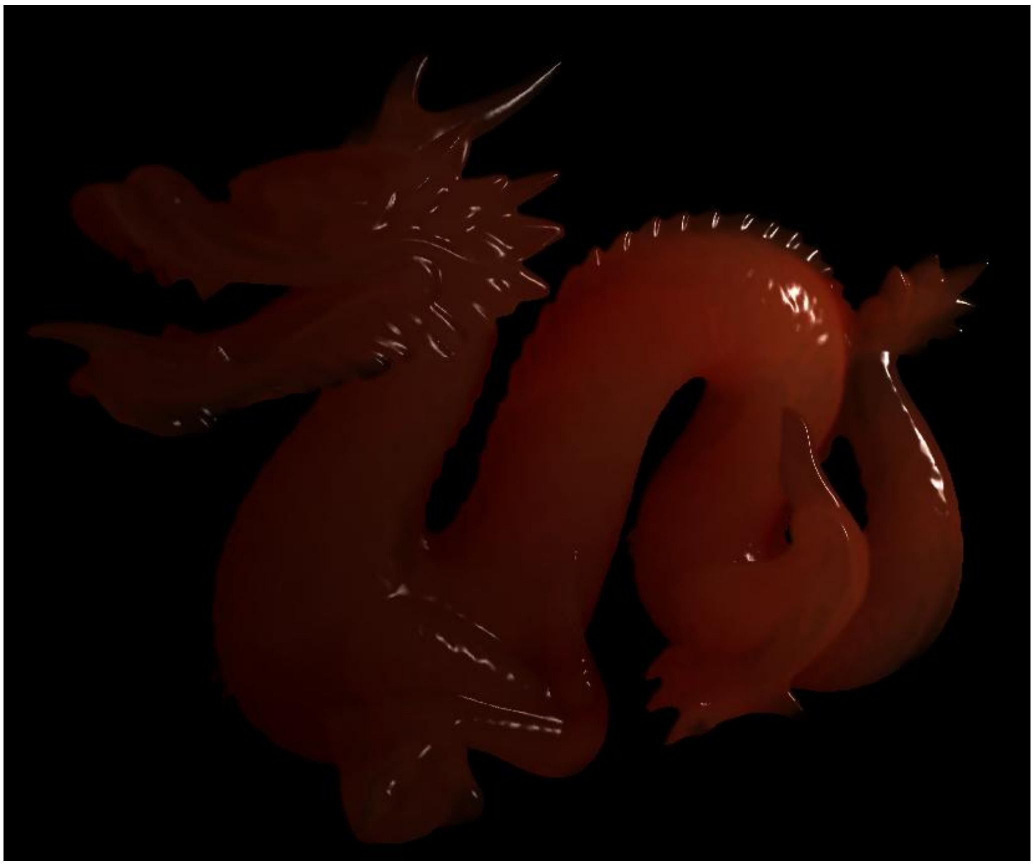


FIGURE 62 - SUBSURFACE SCATTERING THROUGH KETCHUP

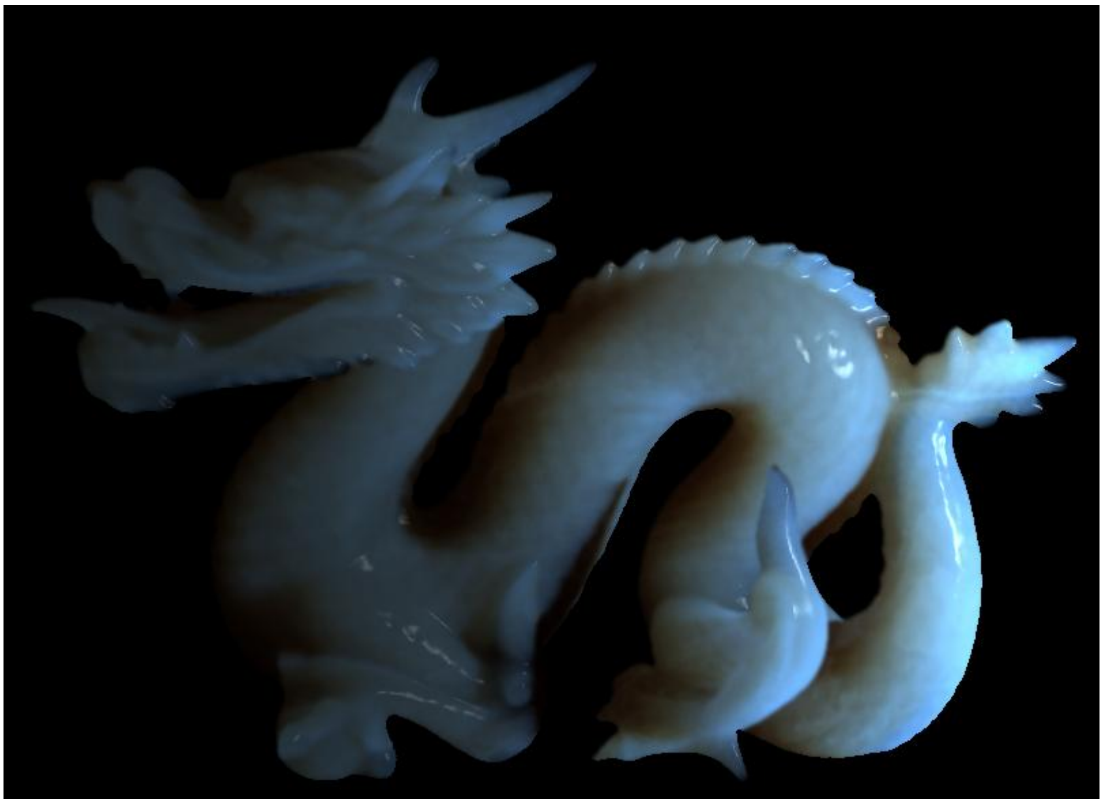


FIGURE 63 - SUBSURFACE SCATTERING THROUGH SKIM MILK

5.3 SKIN RENDERING

All skin rendering results use the Kelemen/Szirmay-Kalos specular BRDF for comparison.

The first image is a rendering using simple Lambertian diffuse reflection for comparison. The skin looks quite unnatural, and hard.

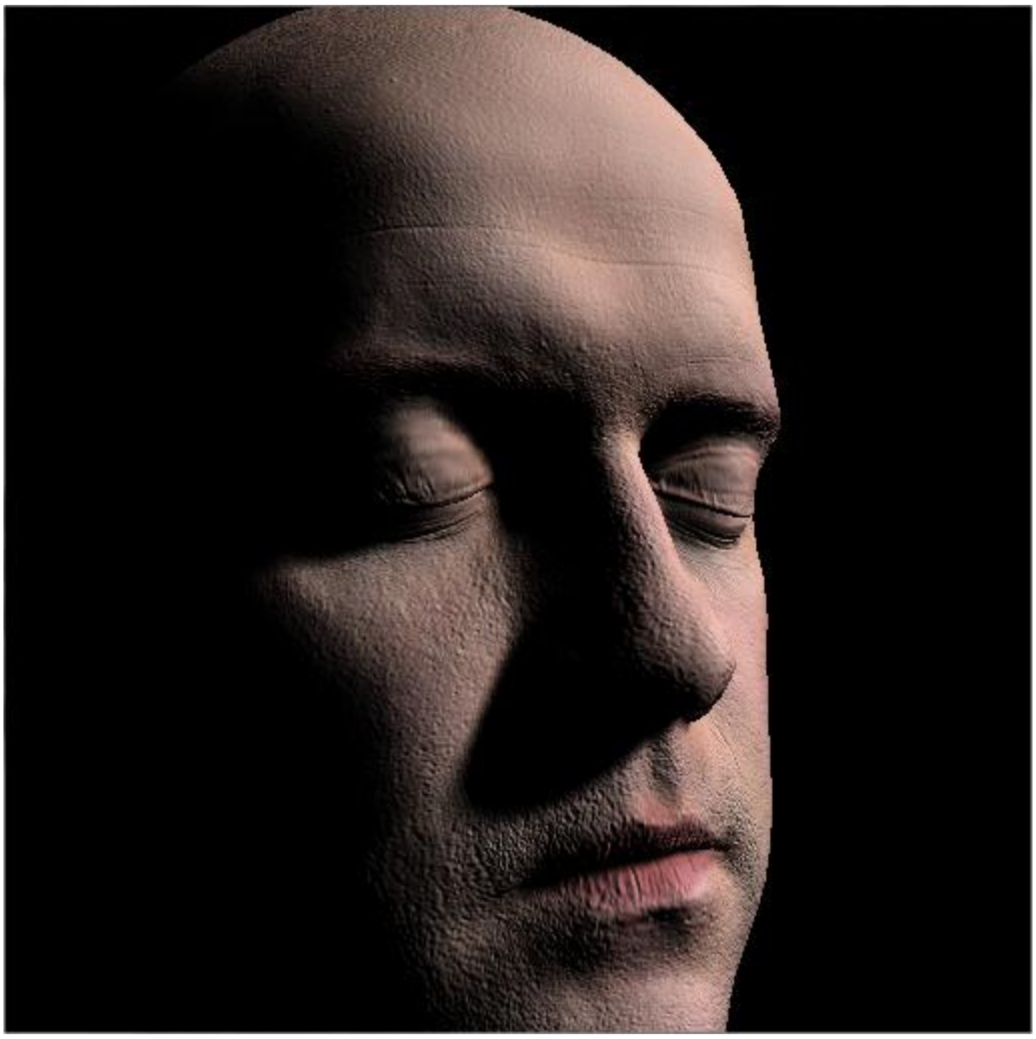


FIGURE 64 - SKIN RENDERING WITH SIMPLE LAMBERTIAN DIFFUSE REFLECTION

Next, the full 5-pass texture space diffusion model implemented after [d'Eon and Luebke 2007]. This model is very complex, but the results are also very good. It has set a gold standard for skin rendering in real-time, but is still too computationally heavy for high performance applications like games.

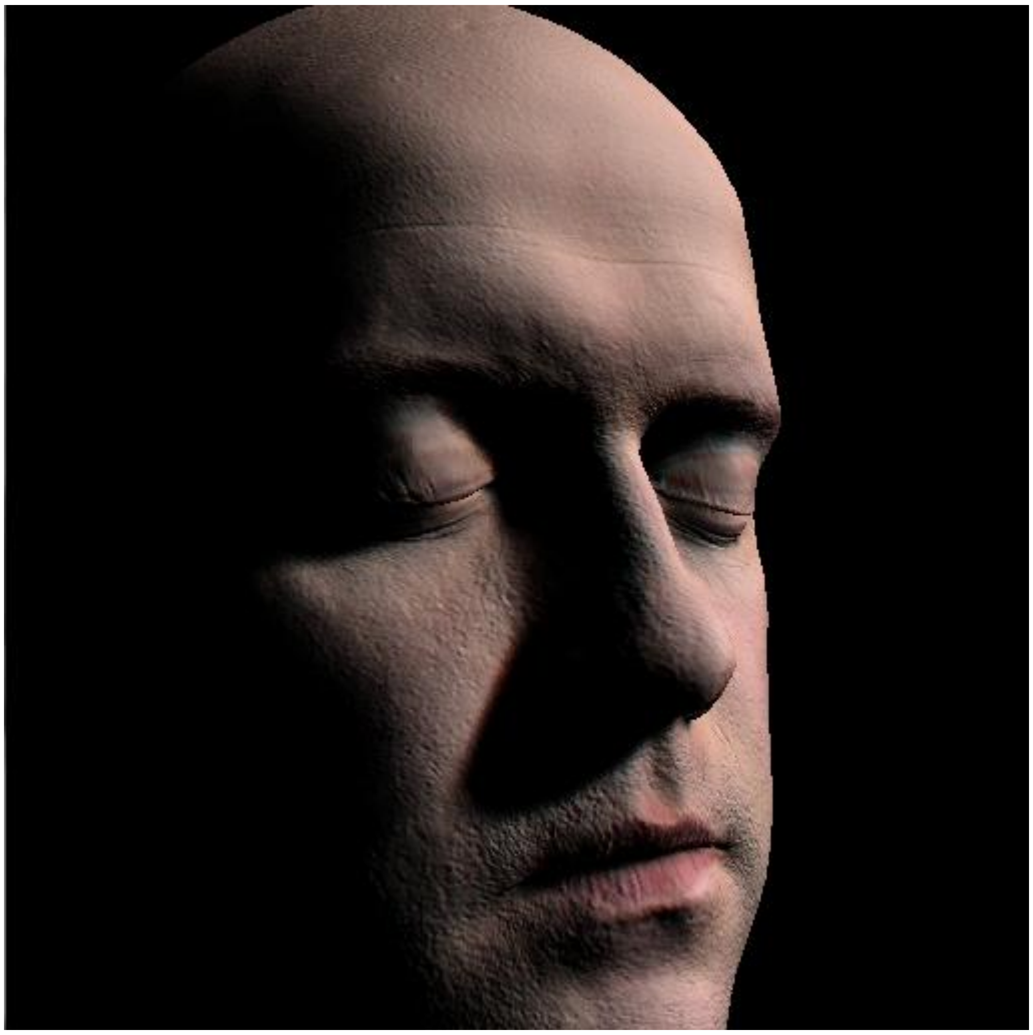


FIGURE 65 - SKIN RENDERING WITH FULL 5-PASS TEXTURE SPACE BLUR

The next image is after implementing [Hable et al. 2009]. This technique also blurs the diffuse light in texture space, but this time using only a single blur, making it much faster. The results are clearly comparable, and in most cases this technique will be preferred based on its greater speed.

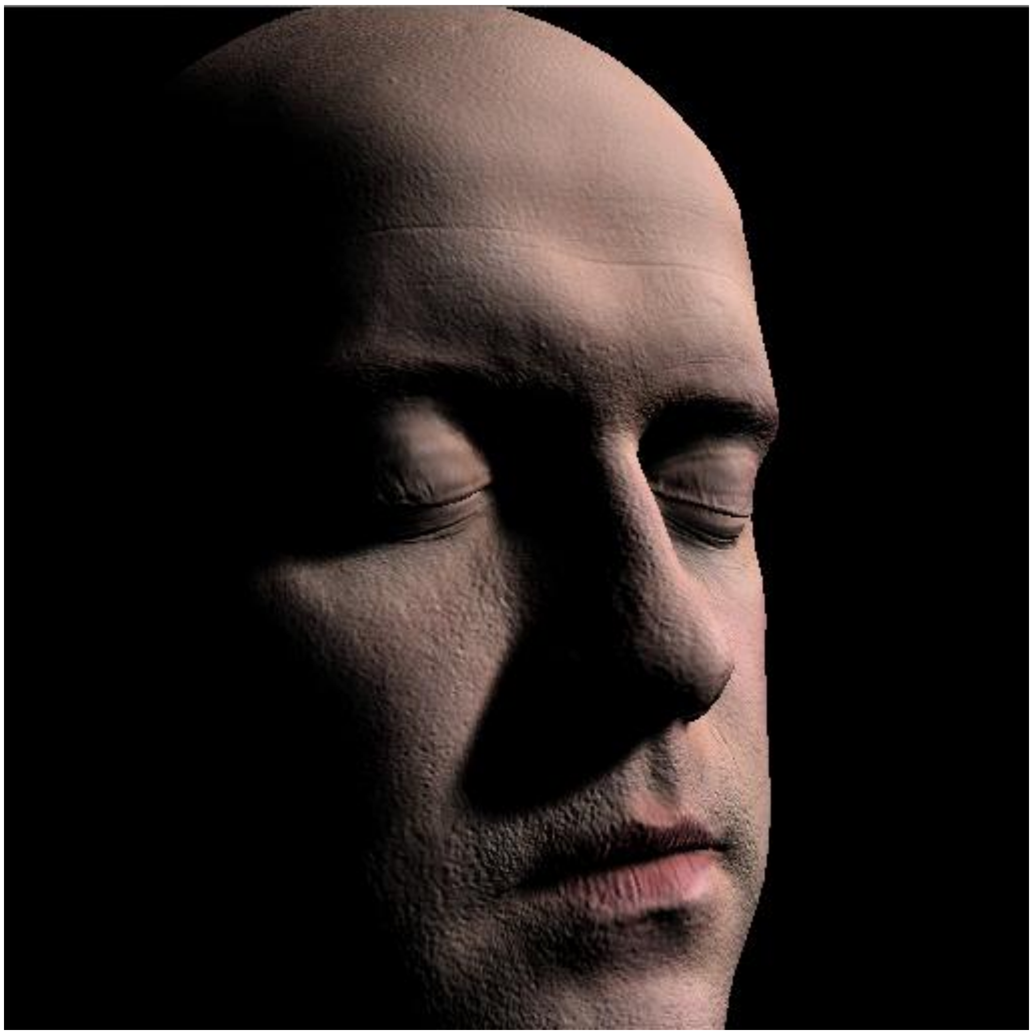


FIGURE 66 - SKIN RENDERING WITH A SINGLE COMBINED BLUR

And finally to show that even a little work helps a lot when rendering skin, a simple screenspace blur using different weights for the different colors. This method is not physically based, but is very fast, and improves a lot upon the standard Lambertian diffuse rendering.

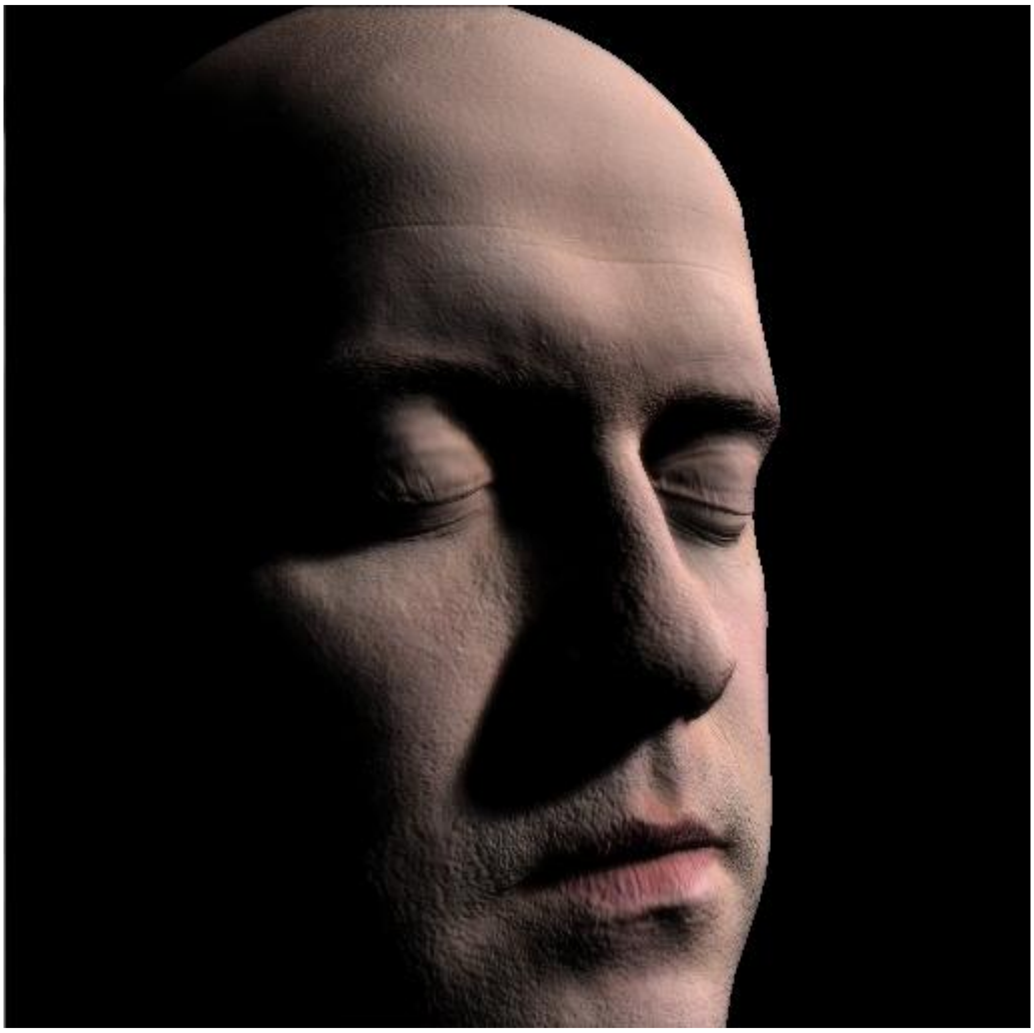


FIGURE 67 - SKIN RENDERING WITH SCREENSPACE BLUR

While the full 5-pass texture space diffusion still gives the best results, it is clear that you can come a long way by just doing something. The one-pass texture space blur, or even a screenspace blur, will do wonders compared to simple Lambertian shading.

6 Conclusion

This thesis set out to evaluate various approaches to real-time rendering of light scattering effects. Methods in the areas of participating media rendering and subsurface scattering have been studied, analyzed and implemented. This chapter summarizes what has been achieved, and covers what is next.

6.1 Participating media rendering

A solution has been implementing using the ray-marching method for rendering of light and shadows through single scattering media. Several optimizations have been analyzed and implemented based on speed, image quality and technique.

Using ideas from various papers, a sample program has been implemented that simulates light scattering in different media, and with different scattering parameters. It has been implemented with a modern graphics engine in mind, and is based on deferred shading, taking advantage of the data available in such an engine.

The implementation supports fog and water, allowing the user to alter the scattering coefficient to emulate any amount of scattering particles in the media. Different scattering models have been implemented supporting primarily forward scattering, back scattering or isotropic scattering. The sample program has support for textured spot lights which are common in deferred render engines. This allows emulation of stained glass windows for example. A simple estimation of heterogeneous media is supported by sampling a Perlin noise texture, giving the appearance of smoke. Finally I have added a multiplication factor allowing the user to enhance the effect of scattering. This is because scattering is inherently a quite subtle effect, and in many cases it will be useful to artificially boost the 'foggyness' of a scene.

6.1.1 Screenspace approach

A sample program has been developed implementing screenspace God rays. This method is based solely on scene the scene depth buffer for shadows, and uses the background texture (in this case a real-time implementation of the sky) for the God rays.

This approach is very simple, but the effect it can have on the overall look of an outdoor scene is quite remarkable. It has several issues, but the computation price is very low, making it very useful when wanting to set the mood of a scene at dawn or dusk for example.

6.2 Subsurface scattering

An interactive solution based on [Ki 2009] has been implemented with support for several different material types. This method leads to very decent results, but is still quite computationally heavy. It is flexible, and being physically based makes it easy to implement several different material types.

6.3 Skin rendering

A sample program has been developed, implementing several approaches to real-time skin rendering. The most realistic is still the one presented in [d'Eon and Luebke 2007], but optimizations from [Hable et al.

2009] give much greater speed while still keeping a very realistic looking skin shading. A screenspace method has also been developed, using a simple diffusion profile. This method is not nearly as correct as the texture space versions, but goes to show that even a little work in subsurface scattering goes a long way towards rendering more believable human faces.

A physically based BRDF has also been implemented based on the Kelemen/Szirmay-Kalos BRDF giving much more realistic specular reflectance than can be obtained by using Phong.

6.4 Improvements and future work

While the solutions presented work very well, there is always room for improvements. Some simplifications would be nice to eliminate, and some methods have inherent flaws that could be improved greatly.

6.4.1 Light propagation volumes

Recently a lot of work has been done in simulating real-time diffuse interreflection. The method developed in [Kaplanyan and Dachsbacher 2010] has great potential for integrating diffuse interreflection with participating media rendering. Another improvement based on light propagation volumes could be the inclusion of multiple scattering and heterogeneous materials as presented in [Engelhardt et al. 2010].

6.4.2 Caustics

Some work is currently being done in the field of real-time caustics, and it could be very interesting to expand these ideas to include participating media. A few methods have been presented in [Liktor and Dachsbacher 2010] and [Hu et al. 2010] but there is still room for improvements.

6.4.3 Multiple scattering and anisotropic scattering

One clear limitation in the work presented in this thesis is the lack of more complex participating media. It could be very relevant to evolve the methods presented to include media like clouds, smoke and fire.

6.4.4 Skin

In [d'Eon and Luebke 2007] the influence of linear space lighting and gamma correction are discussed. The presented solution in this thesis does not include this, and it shows. This is an obvious next step for the solution presented here.

The screenspace solution implemented in this thesis is quite simple, and [Mikkelsen 2010] have presented a better solution using a cross bilateral filter to avoid blurring light over non-touching areas; from the nose to the cheek for example. This is a problem with the presented work, and would be an obvious place to improve.

6.4.5 Subsurface

As the solution presented here is quite slow, it would be interesting to look to the solution presented in [Barré-Brisebois and Bouchard 2011]. While this is a crude approximation, there is potential in combining the technique with the screenspace scattering work done in [Mikkelsen 2010].

Bibliography

Akenine-Möller, T., Haines, E. & Hoffman, N. 2008. *Real-Time Rendering*, (3rd ed.). A K Peters

Andersson, J. 2011. DirectX 11 Rendering in Battlefield 3, *Game Developers Conference 2011*

Barré-Brisebois, C. and Bouchard, M. 2011. Approximating Translucency for a Fast, Cheap and Convincing Subsurface Scattering Look, *Game Developers Conference 2011*

Billeter, M., Sintorn, E. & Assarsson, U. 2010. Real Time Volumetric Shadows using Polygonal Light Volumes, *High Performance Graphics 2010*

Biri, V., Arquès, D. & Michelin, S. 2006. Real Time Rendering of Atmospheric Scattering and Volumetric Shadows. *Journal of WSCG, Vol. 14 (1)* pp65-72

Blinn, J. F. 1977. Models of light reflection for computer synthesized pictures

Borshukov, G. and Lewis, J. P. 2003. Realistic Human Face Rendering for "The Matrix Reloaded", *SIGGRAPH 2003*

Bruneton, E. and Neyret, F. 2008. Precomputed Atmospheric Scattering, *Eurographics Symposium on Rendering 2008*

Børlum, J., Christensen, B. B., Kjeldsen, T. K., et al. 2011. SSLPV: Subsurface Light Propagation Volumes, *High-Performance Graphics 2011*

Chandrasekhar, S. 1950. *Radiative Transfer*

Chang, C.W., Lin, W.C., Ho, T.C., et al. 2008. Real-Time Translucent Rendering Using GPU-based Texture Space Importance Sampling, *Eurographics 2008*

Chen, J., Baran, I., Durand, F. & Jarosz, W. 2011. Real-Time Volumetric Shadows using 1D Min-Max Mipmaps, *Symposium on Interactive 3D Graphics and Games 2011*

Crane, K., Llamas, I. & Tariq, S. 2007. Real-Time Simulation and Rendering of 3D Fluids. *GPU Gems 3*

Dachsbacher, C. and Stamminger, M. 2003. Translucent Shadow Maps

d'Eon, E. and Luebke, D. 2007. Advanced Techniques for Realistic Real-Time Skin Rendering. *GPU Gems 3*

Dobashi, Y., Yamamoto, T. & Nishita, T. 2002. Interactive Rendering of Atmospheric Scattering Effects Using Graphics Hardware. *Graphics Hardware (2002)* pp1-10

Donner, C. and Jensen, H. W. 2005. Light Diffusion in Multi-Layered Translucent Materials, *Proceedings of SIGGRAPH 2005*

Engel, W. 2009. Designing a Renderer for Multiple Lights: The Light Pre-Pass Renderer. *ShaderX7: Advanced Rendering Techniques* pp655-666. Charles River Media

Engel, W. 2003. *ShaderX2*

Engelhardt, T. and Dachsbacher, C. 2010. Epipolar Sampling for Shadows and Crepuscular Rays in Participating Media with Single Scattering, *2010 Symposium on Interactive 3D Graphics and Games*

Engelhardt, T., Novak, J. & Dachsbacher, C. 2010. Instant Multiple Scattering for Interactive Rendering of Heterogeneous Participating Media

Evans, A. and Kirczenow, A. 2011. Voxels in LittleBigPlanet 2, *SIGGRAPH 2011*

Freiburg 2007. Computer Graphics Shadow Algorithms, *Freiburg University Course Notes*

Gautron, P., Marvie, J.E. & François, G. 2009. Volumetric Shadow Mapping, *SIGGRAPH 2009 Talks*

Gosselin, D. 2004. Real Time Skin Rendering, *Game Developers Conference 2004*

Green, S. 2004. Real-Time Approximations to Subsurface Scattering, *GPU Gems*

Hable, J. 2010. Uncharted 2: Character Lighting and Shading, *SIGGRAPH 2010*

Hable, J., Borshukov, G. & Hejl, J. 2009. Fast Skin Shading. *ShaderX7: Advanced Rendering Techniques* pp161-173. Charles River Media

Hargreaves, S. 2004. Deferred Shading, *Game Developers Conference 2004*

Hoffman, N. and Preetham, A. J. 2002. Rendering Outdoor Light Scattering in Real Time

Hu, W., Dong, Z., Ihrke, I., et al. 2010. Interactive Volume Caustics in Single-Scattering Media

Imagire, T., Johan, H., Tamura, N. & Nishita, T. 2007. Anti-Aliased and Real-Time Rendering of Scenes with Light Scattering Effects

James, R. 2003. True Volumetric Shadows. *Graphics Programming Methods* pp353-366

Jarosz, W., Zwicker, M. & Jensen, H. W. 2008. The Beam Radiance Estimate for Volumetric Photon Mapping

Jensen, H. W. and Christensen, P. H. 1998. Efficient simulation of light transport in scences with participating media using photon maps, *SIGGRAPH '98*

Jensen, H. W., Marschner, S. R., Levoy, M. & Hanrahan, P. 2001. A Practical Model for Subsurface Light Transport, *SIGGRAPH '01*

Jimenez, J., Sundstedt, V. & Gutierrez, D. 2009. Screen-Space Perceptual Rendering of Human Skin

Kajiya, J. T. 1986. The Rendering Equation. *SIGGRAPH Comput. Graph., Vol. 20(4)* pp143-150

Kaplanyan, A. and Dachsbacher, C. 2010. Cascaded Light Propagation Volumes for Real-Time Indirect Illumination

Kelemen, C. and Szirmay-Kalos, L. 2001. A Microfacet Based Coupled Specular-Matte BRDF Model with Importance Sampling, *Eurographics 2001*

Keller, A. and Heidrich, W. 2001. Interleaved Sampling

Ki, H. 2009. Real-Time Subsurface Scattering Using Shadow Maps. *ShaderX 7* pp467-478

Liktor, G. and Dachsbacher, C. 2010. Real-Time Volumetric Caustics with Projected Light Beams

Mech, R. 2001. Hardware-Accelerated Real-Time Rendering of Gaseous Phenomena. *Journal of Graphics, GPU, and Game Tools, Vol. 6 (3)* pp1-16

Mikkelsen, M. S. 2010. Skin Rendering by Pseudo-Separable Cross Bilateral Filtering, *Naughty Dog Inc.*

Mitchell, J. 2004. Light Shafts: Rendering Shadows in Participating Media, *Game Developers Conference 2004 Presentations*

Mitchell, K. 2007. Volumetric Light Scattering as a Post-Process. *GPU Gems 3* pp275-285

Mittring, M. 2009. "A bit more Deferred" - CryEngine 3, *Triangle Game Conference 2009*

Nicodemus, F. E., Richmond, J. C., Hsia, J. J., et al. 1977. *Geometrical Considerations and Nomenclature for Reflectance*

Nishita, T., Sirai, T., Tadamura, K. & Nakamae, E. 1993. Display of The Earth Taking into Account Atmospheric Scattering, *SIGGRAPH 1993*

NVIDIA 2008. Volume Light

Pharr, M. and Humphreys, G. 2004. *Physically Based Rendering: From Theory to Implementation*. Morgan Kaufmann Publishers Inc.

Sousa, T. 2008. Crysis Next Gen Effects, *Game Developers Conference 2008 Presentations*

Sousa, T. 2007. Vegetation Procedural Animation and Shading in Crysis. *GPU Gems 3*

Sun, B., Ramamoorthi, R., Narasimhan, S. G. & Nayar, S. K. 2005. A Practical Analytic Single Scattering Model for Real Time Rendering

Tóth, B. and Umenhoffer, T. 2009. Real-time Volumetric Lighting in Participating Media, *Eurographics 2009 Short Papers*

Web - Benjamin Button, *http://www.picknmixflix.com/c/benjamin_button.php*, 26-02-2012.

Web - Hable, *http://filmicgames.com/archives/557*, 27-02-2012.

Web - Infinite Realities, *http://www.ir-ltd.net/*, 11-02-2012.

Web - Lebedev, *http://lebedev.as/index.php?p=1_10_NEW-Articles*, 26-02-2012.

Web - Urbano Álvarez, *http://xnacommunity.codeplex.com/wikipage?title=Componente%20Scatter*, 17-12-2011.

Wenzel, C. 2006. Real-time Atmospheric Effects in Games, *SIGGRAPH 2006*

Wikipedia - BSDF, *http://en.wikipedia.org/wiki/Bidirectional_scattering_distribution_function*, 26-02-2012.

Wikipedia - Gollum, *http://en.wikipedia.org/wiki/Gollum*, 26-02-2012.

Wyman, C. 2011. Voxelized Shadow Volumes, *High Performance Graphics 2011*

Wyman, C. and Ramsey, S. 2008. Interactive Volumetric Shadows in Participating Media with Single-Scattering