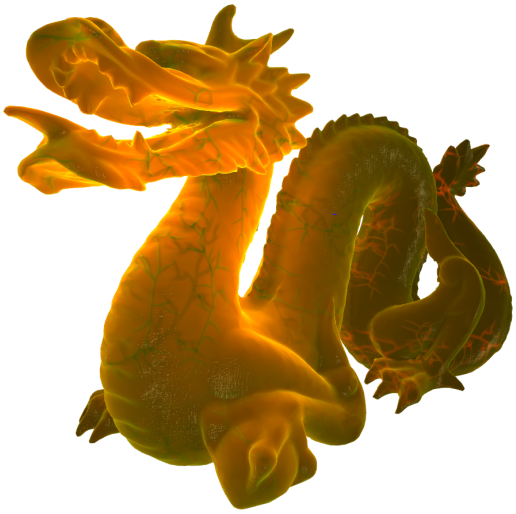
CHALMERS



Realistic rendering of organic materials
in real-time simulations

Master of Science Thesis in Computer Science and Engineering

DAVID NORBERG

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Göteborg, Sweden, January 2010

Realistic rendering of organic materials
in real-time simulations

David Norberg

Examiner: Ulf Assarsson

Department of Computer Science and Engineering
Chalmers University of Technology
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Department of Computer Science and Engineering
Göteborg, Sweden January 2010

Abstract

This report presents a case study of two techniques for rendering organic materials. Both techniques are examples of subsurface scattering algorithms. The first approximates single scattering in a multilayered material by ray marching and sampling a heightfield texture with the layers' depth information, while the second method approximates multiple scattering by using a blurred lightmap and multiple texture layers. The result of the implementations are given and both of the techniques suitability for the application of surgical simulation is ascertained.

Preface

This report describes the result of a masters thesis for the Computer Science and Engineering department at Chalmers University of Technology (CTH). The work was carried out at Surgical Science AB in Gothenburg.

I would like to thank my supervisor Anders Larsson at Surgical Science, my examiner Ulf Assarsson at CTH and everyone else at Surgical Science for their help, inspiration and good company.

Contents

1 Introduction

The purpose of simulation software is to simulate some aspects of the real world. In surgical simulation the goal is to replicate the environment of the human body in detail. Good rendering techniques for such organic materials are needed to achieve a realistic image.

When rendering hard surface materials such as plastic, traditional lighting techniques such as bump mapping give quite realistic results. Translucent materials on the other hand have a much smoother look due to light interactions inside the material and can therefore not be accurately rendered using traditional techniques.

Since this thesis has been conducted in the interest of realistic surgical simulation, this report focuses on organic materials and two rendering techniques for rendering them are studied. Organic materials exhibit a few defining properties that give them their characteristic look. The two rendering techniques try to emulate these to different degrees.

1.1 Properties of translucent materials

When light hits a non-translucent surface it is either absorbed or scattered off the surface from the point of impact in some direction. When light hits translucent material it may also scatter into the surface. Once inside the material, light can scatter several times before it is either absorbed or it exits to the surface (possibly at some other point than it entered). This process tends to spread out the light inside the material and gives a less hard and smoother look to the surface.

In non-homogeneous translucent materials such as biological tissue there are also important visible details beneath the surface. Veins and arteries can be seen trough human skin. When looking at subsurface details from different angles, the details' change of position tends to be different depending on how deep the it lies inside the material. This is called depth parallax and is a important property to simulate since it gives the material its sense of depth.

When light enters the material different wavelengths of the light are absorbed depending on the material properties. Human skin for instance scatters mostly red wavelengths, which is why your hand glows red if you put it in front of a lamp [Guillaume et al 2006].

When light scatters within a medium it is useful to distinguish between single and multiple scattering. Single scattering is when the light only scatters once (or only a few times) within the medium before exiting. Multiple scattering is when the light scatters many times before exiting. The two different types of light scattering produce different kinds of effects. Single scattering tends to give more high frequency (sharper) effects like volumetric caustics, while multiple scattering often acts as a low-pass filter and blurs the lighting [Walter et al. 2009]. Simulating these different kinds of effects are important in order to give a realistic appearance.

1.2 Related work

A lot of work has been done in the field of subsurface scattering, both for real-time and offline rendering. Offline rendering techniques are often quite computationally expensive and are often based on physically accurate models, for example [Jensen 2001],[Jensen and Buhler 2002] and [Donner and Jensen 2005]. A big research area within subsurface scattering is realistic rendering of human skin (especially faces). [Borshukov and Lewis 2003] is a good example. Much research has also

been done for skin rendering in real-time applications, for example [Gosselin 2004] and [d'Eon and Luebke 2007]. Because of its computational expense, subsurface scattering algorithms use lots of approximations for real-time applications, but can nevertheless deliver some good results. The next section will introduce the two techniques studied in this report.

1.3 Two techniques

In this thesis, two different techniques will be evaluated as candidates for modeling organic materials. The first technique called subsurface texture mapping ([Guillaume et al 2006]) uses a more advanced lighting model but is still possible to run at interactive speeds on modern graphics hardware, while the other technique ([Oat 2006]) skips correctness and tries to emulate some of the properties of organic materials by combining a few well established techniques that run really fast.

2 Theory

This section describes the theory of [Guillaume et al 2006] and [Oat 2006]. Both techniques use the concept of multiple layers in their design to achieve a description of subsurface details at different depths, altough they go about it in different ways.

2.1 Subsurface Texture Mapping

This section goes through the theory of the subsurface texturing method. It uses [Guillaume et al 2006] as a reference article. The implementation specific details, practical problems and solutions are presented in section 3.

2.1.1 Overview

This is the most physically correct approach of the two. It uses multiple layers to model the subsurface details. Instead of using the simpler approach of modeling layers of constant thickness this technique uses layers of variable thickness. This is accomplished by representing each layer with a heightfield texture. The method builds on the technique of relief mapping [Policarpo et al. 2005], where surface details are modeled by ray-intersection of a similar heightfield. Unlike relief mapping however, which models surface details, this method instead models the layers beneath the surface. Also, unlike relief mapping, there is no need to find the exact intersection point of the height map. Instead, the ray is sampled at regular intervals, and the different heightfields are used to distinguish in which layer the current sample point is located.

The subsurface texture mapping algorithm simulates single scattering. The basic idea of the algorithm is illustrated in Figure 1. The viewer is located at point C and looks at the material at the surface point P. A ray is cast in the viewing direction and sampling occurs at points M at regular steps along the ray from point P to point M_{max}. We are also interested in the depth of the materials at point M in regards to the light source S. This depth is the length of the line KM from a point K at the surface and down the light direction to M.

Point M's single scattering contribution, as seen from point P is expressed as a function $L_M(P, \omega_{out})$, where ω_{out} is the direction to the viewer.

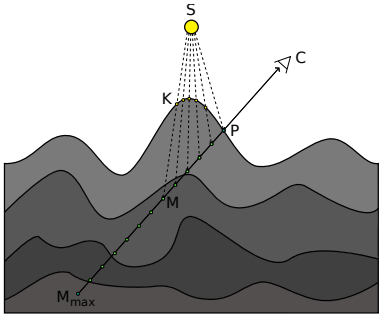


Figure 1: An illustration of the algorithm.

$L_M(P, \omega_{out})$ can be defined as:

$$L_M(P, \omega_{out}) = Q(M, \omega_{out}) e^{\int_M^P -\sigma_t(s)ds} \tag{1}$$

$$Q(M, \omega_{out}) = \sigma_s(M) p(M, \omega_{out}, \omega_{in}) L_{ri}(M, \omega_{in}) \tag{2}$$

These functions are most easily understood by examining each part separately. The expression $e^{\int_M^P -\sigma_t(s)ds}$ models the attenuation of scattered radiance due to absorption and scattering along the viewing ray between points M and P. Scattered light closer to the surface from P's point of view, will give a greater contribution than light deeper into the material. This term is then used to scale the single scattering result, $Q(M, \omega_{out})$ of point M.

$Q(M, \omega_{out})$ can also be divided into several contributing parts. $\sigma_s(M)$ is the scattering coefficient, which is a scaling factor that affects the strength of the contribution of point M. $\sigma_s(M)$ can be different for each layer and may also vary with color (This makes it possible to model different behaviour for different wavelengths of light).

The function $p(M, \omega_{out}, \omega_{in})$ is the phase function and it models how light coming from the direction ω_{in} scatters in the direction ω_{out}. Different phase functions are available but the one used here is the Schlick phase function:

$$p(\theta) = \frac{1 - g^2}{4\pi(1 + g\cos(\theta))^2} \tag{3}$$

where θ is the angle between ω_{in} and ω_{out}. The parameter g is called the average cosine and models the degree of anisotropy of the phase function. Changing the g parameter can change how broad or narrow the scattering is and its general direction (toward or away from the viewer).

The final part of $Q(M, \omega_{out})$ is the reduced intensity $L_{ri}(M, \omega_{in})$, this is where the distance $|KM|$ is needed to calculate the light attenuation at the depth of M.

The function $L_M(P, \omega_{out})$ is only the contribution of a single sample point M. In order to get all the scattered light arriving at point P all sample contributions need to be added together:

$$L(P, \omega_{out}) = \sum_P^{M_{max}} L_M(P, \omega_{out}) \delta_M \tag{4}$$

$$= \sum_P^{M_{max}} Q(M, \omega_{out}) e^{\int_M^P -\sigma_t(s)ds} \delta_M \tag{5}$$

where δ_M is the distance between each sample point along the viewing ray. Because only a few points between P and M_{max} are actually sampled, the attenuation term $e^{\int_M^P -\sigma_t(s)ds}$ need to be discretized. This formula can be rewritten as:

$$e^{\int_{M_{N+1}}^{P} -\sigma_t(s)ds} = e^{\int_{M_{N+1}}^{M_N} -\sigma_t(s)ds + \int_{M_N}^{P} -\sigma_t(s)ds} \tag{6}$$

$$= e^{\int_{M_N}^{P} -\sigma_t(s)ds} e^{\int_{M_{N+1}}^{M_N} -\sigma_t(s)ds} \tag{7}$$

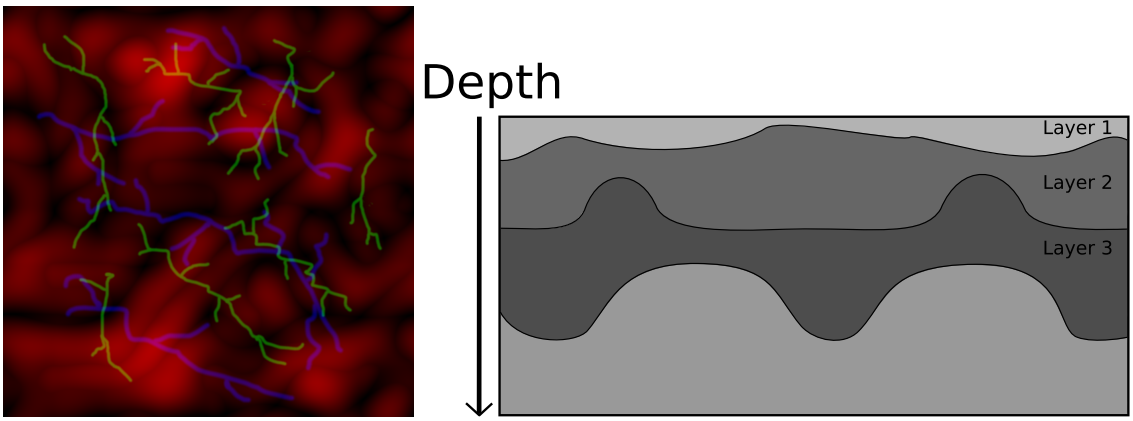
In discretized form this becomes:

$$e^{\sum_{P}^{M_{N+1}} -\sigma_t(s)\delta_s} = e^{\sum_{P}^{M_N} -\sigma_t(s)\delta_s} e^{-\sigma_t(M_{N+1})\delta_{M_{N+1}}} \tag{8}$$

where δ_s is the length of the step along the view vector.
So by using the attenuation term of the previous sample N, we can calculate the attenuation for $N+1$. This is very practical since the raymarching algorithm goes through the points one at a time, starting at P. Since the $\sigma_t(s)$ factor may vary for each layer, it is necessary to know in which layer the point is located. This will be explained in greater detail later on.

2.1.2 Modeling the layers

In order to have an efficient implementation of the layers, they are represented by heightmap textures. A good way to use the graphics hardware efficiently is to store the heightmap of each layer in a color channel of a RGBα texture. This way it is possible to model 4 layers per texture.



(a) (b)

Figure 2: The subsurface texture is shown in 2(a), only the RGB channels are visible. In 2(b) the layers represented by 2(a) are illustrated from the side.

An example of a subsurface texture can be seen in Figure 2(a). The different colors represent the different layers, with red representing a layer of larger irregular bumps, blue represents a layer of pretty thick veins and green a layer of thinner veins.

Figure 2(b) shows a representation of these layers from the side. It is useful to understand how these heightmaps are eventually used. A single channel of RGBα texture can represent values from 0 to 255 or 0.0 to 1.0 as a floating point value. It is important to decide what these channels actually represent.

When modeling a layer that begins at depth of 1.0, it can be represented either with the color white (1.0) or black (0.0). Given the texture in Figure 2(a) the green

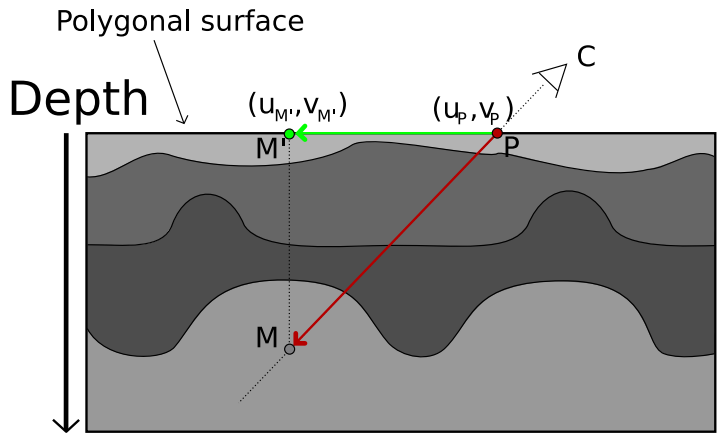


Figure 3: Finding the texture coordinate of M.

channel could either be visible as veins sticking out of the material or dug into the material.

The layer information is ultimately used to check which layer a sample point M is located. This look-up is illustrated in Figure 3. In order to find the layer where M is located, one has to find the texture coordinate which describes the layer depths of that point. As can be seen in Figure 3, finding the correct layer is just a matter of comparing the depth of point M with the layer depths sampled from the texture.

The vector PM in Figure 3 needs to be transformed into tangent space in order for any texture comparisons to make any sense. Tangent space is a local coordinate system for each vertex on a mesh. The depth vector of tangent space corresponds to the geometric normal of the vertex, and the tangent and bi-normal vectors form a plane that is perpendicular to the normal and should be related to the texture space axis s and t. Because of the tangent space vectors correspondence with texture space, they can be used to calculate new texture coordinates.

The steps needed to find the correct layer are:

- Calculate the vector PM, which is a vector along the viewing ray from point P to point M.
- Project PM into tangent space. Since the goal is to find the correct texture-coordinate, the tangent space depth component is not needed.
  $(ds, dt) = (PM \bullet T, PM \bullet B)$
  where the $\bullet$ operator is the dot product and T and B are the tangent and bi-normal vectors. If the tangent space vectors are correctly defined, then the vector (ds, dt) is a 2D vector in texture space.
- The texture coordinate (U_P, V_P) of point P are known and can now be combined with the vector (ds, dt) to form the new texture coordinate at M':
  $(U'_M, V'_M) = (U_P, V_P) + (ds, dt)$
- Compare the depth of M with the depths stored in the subsurface texture at coordinate (U'_M, V'_M) and determine the correct layer.

As can be seen in equation (2) in section 2.1.1, an estimation of the lights reduced intensity at point M is needed for the scattering computation. This is the topic of the next section.

2.1.3 Estimating the reduced intensity

To compute the reduced intensity $L_{ri}(M, \omega_{in})$ (see Equation 2) it is necessary to know the distance $||KM||$ from a point K on the object surface and along the light-ray to point M (see Figure 1).

The reduced intensity equation can be written as:

$$L_{ri}(M, \omega_{in}) = L_i(K, \omega_{in})e^{-\sigma_t||KM||} \tag{9}$$

where $L_i(K, \omega_{in})$ is the light intensity at point K on the surface. As can be seen in this equation, the distance $||KM||$ together with the extinction coefficient σ_t decides how much the light is attenuated.

The distance $||KM||$ can be accurately calculated by finding the intersection of the ray SM with the surface (where SM is a ray from the light source S to point M)). However, since the light source can move around, the intersection point K can be anywhere on the surface of the model and it is infeasible to easily find such general intersections with standard graphics hardware. An approximation of $||KM||$ is needed.

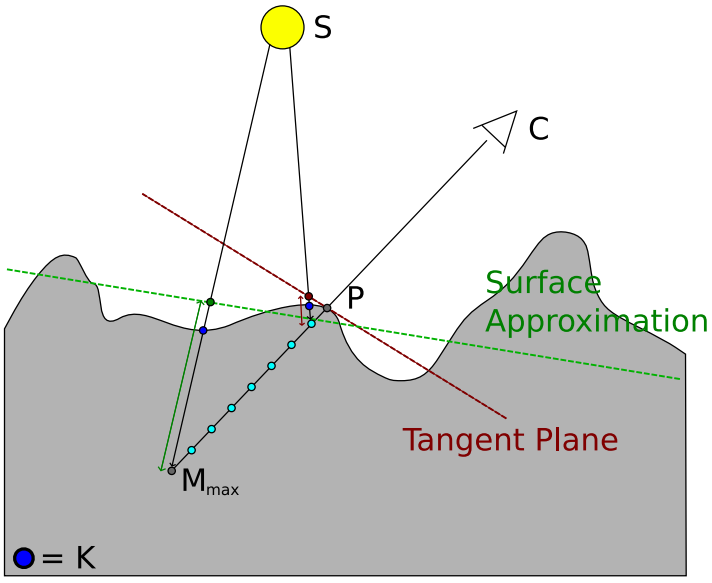


Figure 4: An overview of the planar approximations used for the calculation of $||KM||$

One way to approximate $||KM||$ is to use planar approximations of the geometry. The idea is illustrated in Figure 4. For sample points near the point P on the surface, the light tends to strike the surface near P and can therefore be approximated by P:s tangent plane (the red line in Figure 4). For sample points deeper within the medium however, the light is more likely to strike further away from P and thus making the tangent plane a poor approximation. It is therefore a good idea to introduce another plane to approximate the surface for these points.

In order to decide how large an area of the surrounding surface should be approximated, the concept of light attenuation is used. Since the light will have negligible impact after a certain distance, it is unnecessary to approximate the geometry for light which reaches further away. This is illustrated in Figure 5.

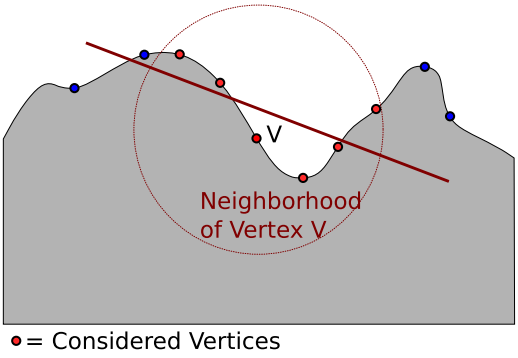


Figure 5: The planar approximation is calculated from points within a certain radius of P. Light reaching outside of this radius is considered to give negligible contribution.

Given the mediums extinction coefficient, one can compute a radius where the light is considered to give good contributions. By picking all the points within this radius from P and using them in a least squares fitting method it is possible to find a good planar approximation of the surface. The plane is then the best fitting planar description of the chosen points. This plane is illustrated in Figure 4 as the green line. This plane is a better approximation for deeper points such as M_{max}.

In order to get a smooth transition between the planes, as well as better approximations for intermediate points (points between M_{max} and P), linear interpolations of the two planes are calculated. The point M_{max} uses the new calculated plane and P uses it's tangent plane. The intermediate points each use a plane that is a linear interpolation between the least squares plane and the tangent plane.

A plane can be represented by a normal and a point that lies on that plane. Point P and it's normal forms one such plane and the least squares method produces another. These planes are used to form several new planes that are good surface approximations for each sample point. The linearly interpolated plane associated with sample point M is denoted $\Pi_M\{\vec{N}_M, P_M\}$. where $\vec{N}_M$ is the plane normal and P_M is a point on the plane.

These planes are calculated as:

$$\Pi_M : \begin{cases} \vec{N}_M &= \frac{1}{\alpha+\beta}[\alpha\vec{N}_P + \beta\vec{N}_{M_{max}}] \\ P_M &= \frac{1}{\alpha+\beta}[\alpha P_P + \beta P_{M_{max}}] \end{cases} \tag{10}$$

where $\alpha = ||MM_{max}||$ and $\beta = ||MP||$, that is the distance between M and M_{max} and M and P, respectively. The normal $\vec{N}_P$ is the tangent planes normal while $\vec{N}_{M_{max}}$ is the normal of the best fit plane (which is used when sampling at M_{max}), and similarly for the points P and M_{max}.

Figure 6 illustrates the calculation of the distance $||K'M||$ which is an approximation to the real distance $||KM||$. The distance h can be calculated as $h = |P_M M \bullet N_M|$. This dot product can be seen as projecting the vector $P_M M$ on the plane normal N_M. Then h is the length of this projection. One can obtain the distance $||K'M|| = \frac{h}{\cos\theta}$ by the definition of cosine (the ratio between the side closest to the angle and the hypotenuse). $\cos\theta$ can be obtained by the dot product between the light direction ω_{in} and the plane normal N_M, that is $\cos\theta = \omega_{in} \bullet N_M$.

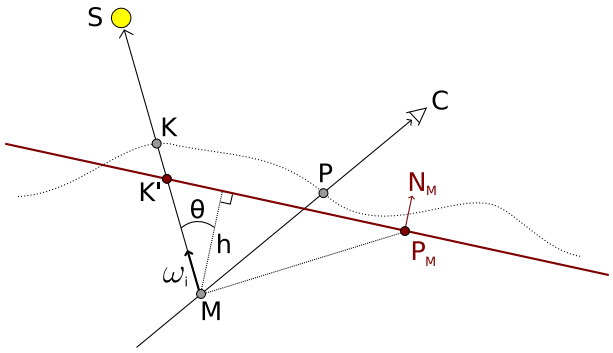


Figure 6: The approximate distance $||K'M||$ is calculated with the surface plane approximation Π_M.

Putting all of these together yields:

$$||K'M|| = \frac{|P_M M \bullet N_M|}{\omega_{in} \bullet N_M} \tag{11}$$

In order to get a correct estimate of the reduced intensity it is necessary to take the different layers into account. The formula for the reduced intensity of a three layer material is given by:

$$L_{ri}(M, \omega_{in}) = L_i(K, \omega_{in}) e^{-\sigma_t^1 d_1} e^{-\sigma_t^2 d_2} e^{-\sigma_t^3 d_3} \tag{12}$$

where d_i is the distance travelled by the light in layer i and σ_t^i is the extinction coefficient of the i:th layer.

Having obtained the approximate total distance the light travels within the medium to point M, it is possible to make an estimate of the distances the light travels in each of the layers before reaching M. Ideally d_i would be calculated by finding the intersection of the layers and the light ray, which would take a lot of texture lookups, but due to performance considerations it is necessary to make a fast estimate instead.

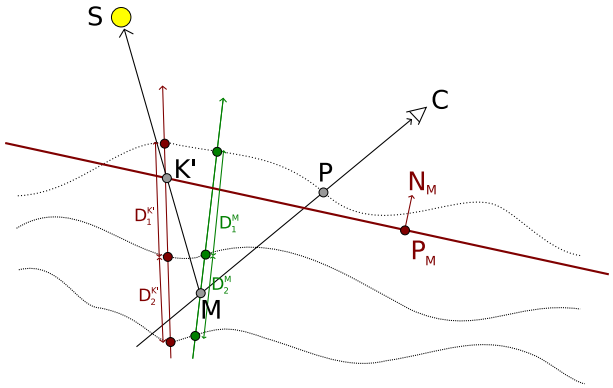


Figure 7: The thickness of the different layers are estimated to find the distance the light travels in each layer.

An approximate layer thickness is obtained by taking the average value of the subsurface map at the texture coordinates of K' and M.

$$D_j = \frac{D_j^M + D_j^{K'}}{2} \tag{13}$$

where D_j^M is the depth of the j:th layer at M and $D_j^{K'}$ is the depth of the j:th layer at K' (see Figure 7). The layers are therefore approximated by a constant depth for each sample point M, which leads to a simple calculation of d_i[1]:

$$d_i \approx \min\left(\max\left(||K'M|| - \sum_{j=0}^{i-1}\frac{D_j}{\cos(\theta)}, 0\right), \frac{D_i}{\cos(\theta)}\right) \tag{14}$$

2.1.4 The algorithm

The following points give an overview of the operation of the algorithm:

- Initialise vectors and variables:
  $Attenuation_{PM} = 1.0$
  $currentLayer = 0$
  $M_{max} = \frac{P-(Depth_{max}*\omega_{out})}{N_P \bullet \omega_{out}}$
  $PM_{step} = \frac{PM_{max}}{numberofsamples}$
  $(U_M, V_M) = (U_P, V_P)$
  $depth_M = 0$
  $depthStep = \frac{Depth_{max}}{numberofsamples}$

- project PM_{max} into tangent space:
  $(ds, dt) = (PM_{max} \bullet T, PM_{max} \bullet B)$
  $textureCoordinateStep = \frac{(ds,dt)}{numberofsamples}$

- For each sample point M:

  - Localise in which layer M is located.
  - Estimate the reduced intensity $L_{ri}(M, \omega in)$.
  - Calculate the single scattering at M:
    $L(M, \omega_{out}) = \sigma_s^{CurrentLayer} * L_{ri}(M, \omega in) * p^{currentLayer}(\omega_{in} \bullet \omega_{out})$
  - Add the attenuated scattering contribution of M to the total scattering at point P:
    $L(P, \omega_{out}) += L(M, \omega_{out}) * Attenuation_{PM} * ||PM_{step}||$
  - Step all variables to next sample M:
    $(U_M, V_M) += textureCoordinateStep$
    $depth_M += depthStep$
    $M += PM_{step}$
    $AttenuationPM *= e^{-\sigma_t^{currentLayer}||PM_{step}||}$

- return $L(P, \omega_{out})$

Notice how the sample point start out with the same values of P and is incrementally stepped along the viewing ray. The reduced intensity calculation is performed as described in section 2.1.3. The T and B variables are the tangent and bi-normal vectors of tangent space.

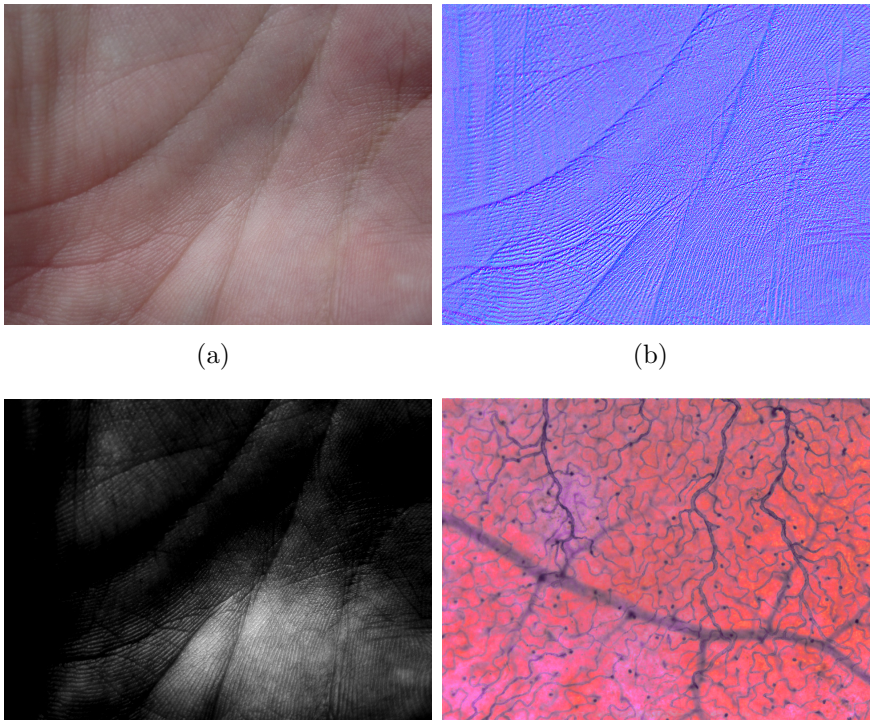
[1]This formula differs a bit from the one given in [Guillaume et al 2006], this is explained in section 3.

2.2 Multitexturing with depth parallax

Compared to the technique presented in the previous section, the following technique is a lot less physically correct. Instead it tries to cheat by mimicking a few perceptual properties of translucent multilayered materials. This theory section is basically a summary of the article by [Oat 2006] and will not be expanded in this section. A few additions to the article will be presented in section 3. The following sections will go through each part of the algorithm followed by a description of how all of them fit together.

It is first useful to get an idea of how the layers are represented by different types of textures. The outer layer of the material use a base texture as its diffuse component and a normal map for specular lighting. It also has a opacity map to model different levels of transparency along the surface. The inner layer (or layers), do not have a normal map but a base texture. Should there be more than one inner layer, the intermediate layers may also use opacity maps.

An example of textures for a two layer material is given in Figure 8. The image in Figure 8(a) is a close-up photograph of skin, 8(b) shows a normal map of the skin (contrast enhanced in order to make the details clearer), 8(c) shows an opacity (alpha) map of the skin and 8(d) shows the inner layers base texture.



(a) (b)

(c) (d) Image from [Kolb et al.]

Figure 8: An example of textures used for two different layers.

The layers are linearly interpolated together based on the alpha channel from the opacity map. The Opacity map in this example was created from the base texture photograph with simple filters in GIMP (The GNU Image Manipulation Program) and the normal map was created with a GIMP-plugin. The lightning of the layers will be discussed in later sections.

2.2.1 Depth Parallax

Just linearly interpolating the layers together would make the material look quite flat and not give any impression of depth. To remedy this, one could use what is called depth parallax. Parallax is the apparent shift in position of an object as seen from two different points (that are not on the same line of sight). A good example of parallax is to simply hold out you're hand in front of you're eyes and compare the views of the left and right eye. Figure 9 gives a illustration of how this is done in case of a two layer material.

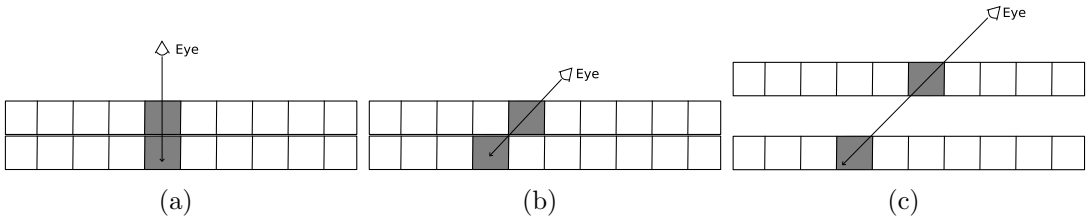


Figure 9: An illustration of how the lower layer texture is accessed depending on viewing position and depth. Figure 9(a) viewing from above. Figure 9(b) viewing from the side, notice how the lower layer texture coordinates change. Figure 9(c), viewed from the side and with greater depth, notice how the texture coordinate shift is more pronounced than in Figure 9(b).

The texture coordinate of the lower layer is shifted depending on how the view-ray "intersects" the lower layer. Adding a distance between the two layers creates a greater depth and changes the amount of shifting of the lower layer. The calculation of the lower layer texture coordinate is shown in Figure 10. The vector of interest here is the transmission vector, which is a vector that points from the outer layers surface point to the inner layers surface point. The calculation of the transmission vector is done in tangent space:

$$\vec{R} = -\vec{V} - 2 * (-\vec{V} \bullet \vec{N}) * \vec{N} \tag{15}$$

$$\vec{T} = < \vec{R}_x, \vec{R}_y, -\vec{R}_z > \tag{16}$$

where $\vec{V}$ is the view vector and $\vec{N}$ is the normal (from a Normal map). The layers are assumed to be flat and parallel in tangent space, which makes expensive intersection testing unnecessary. The first step is to calculate the reflection vector $\vec{R}$ by

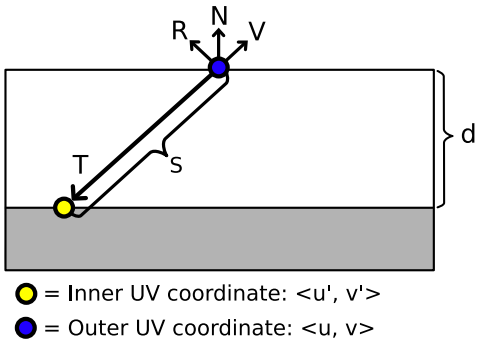


Figure 10: The calculation of the transmission vector.

reflecting the view vector $\vec{V}$ about the normal $\vec{N}$. Notice that since the normal $\vec{N}$

is taken from a normal map, it may not be perpendicular to the surface plane. The transmission vector cannot simply be calculated by inverting the view vector, which would work if the geometric normal was used (as in Figure 10). The next step is to reflect the reflection vector about the surface plane. Since the calculation is done in tangent space, finding the transmission vector is simply a matter of inverting the $\vec{R}_z$ component of the reflection vector. The inner layers new texture coordinates are given by:

$$s = d/|\vec{T}_z| \tag{17}$$

$$< u', v' > \ = \ < u, v > + s < \vec{T}_x, \vec{T}_y > \tag{18}$$

where s is the distance along $\vec{T}$ to the inner layer and d is the distance between the layers. The distance s combined with the transmission vectors x and y components gives a UV offset for from the outer layers texture coordinate to the inner layers texture coordinate.
One problem with this calculation is that the transmission vector can point very far away when the material is viewed from grazing angles. The transmission vector could be almost parallel with the surface layer, which would result in a very extreme texture coordinate shift of the inner layer. In order to make this effect less noticeable, it is a good idea to choose to render the outer layer at grazing angles instead. This is somewhat reminiscent of how the Fresnel effect works on water surfaces, where you can see through the water when looking from above but not from the side.

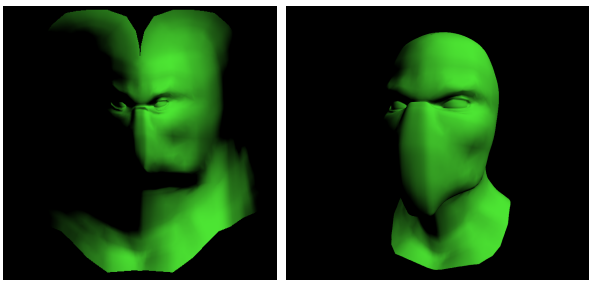
The following term can be used in the linear interpolation of the layers to get this effect:

$$NdotV = saturate(N \bullet V) \tag{19}$$

where N is the normal and V is the view vector. The *saturate* function clamps the value between 0.0 and 1.0. The idea is that you get a scale from 0.0 to 1.0 of how much you should see trough the material. When the view vector and normal coincide, the angle between them are zero and since both are of unit length the value of the dot product becomes $\cos(0) = 1.0$. This should be interpreted as full see through of the material. When viewed from grazing angles however, the angle between the normal and viewvector would be close to 90°, resulting in a dot product close to 0.0 (no see trough).

2.2.2 Approximating light diffusion

When light hits the surface of a translucent material some of it will scatter directly towards the viewer and some will scatter into the material. The light that scatters directly toward the viewer is easy to calculate. It is simply done by using a standard diffuse lighting term as well as a specular term calculated with normals from a normal map. The inner layer(s) however need a different treatment. The light that scatters around inside the material tends to give a blurry look to the inner layer as well as having the light spread out more evenly. This is the look that needs to be approximated. This approximation can be done by rendering the diffuse lighting of the surface into an offscreen texture and then blurring it. The blurring would have the effect of spreading out the lighting a bit and making it softer.
The desired offscreen texture should be used to light the same areas of the model as normal diffuse lighting would. This is accomplished by using the texture coordinates



(a) (b)

Figure 11: Figure 11(a) depicts a lightmap with diffuse lighting (as well as a green colour) for a ninja head model. Figure 11(b), the ninja head rendered with the light map in 11(a).

for the vertices in the vertex shader instead of the usual vertex coordinates. This way the model is "wrapped out" into a 2D representation of the model in texture space. Note that this puts some demands on the model. In order to achieve correct lighting, unique texture coordinates for each vertex is needed. Considering that the model is basically turned into two dimensions there is no longer any depth to test in order to find the correct rendering order of the triangles, which means that overlapping triangles might be rendered incorrectly.

This switch between vertex coordinates and texture coordinates is the only difference from rendering the model in 3D. The light vector is computed from the normal 3D vertex positions, which means the diffuse lighting equation $(N \bullet L)$ can be used as usual.

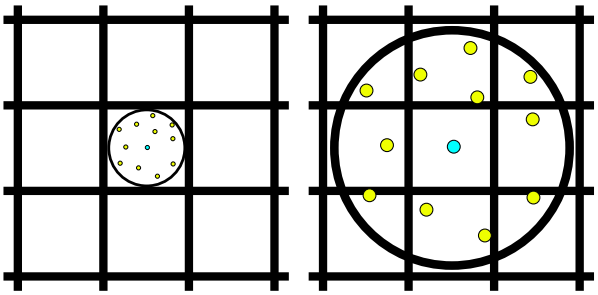
The idea with the diffuse lightmap is that it is to be used as light input for the inner layers. This light scatters around in the material causing it to look smoother and spread out, which is simulated by blurring the lightmap.

Blurring can be accomplished in the pixel shader. For each texel in the image, samples of nearby texels are taken and added together and then divided by the number of samples used. This gives each texel an average colour that depends on the area that was sampled, which gives a blurry look. The sampling can be done in many different ways and in a different radius from the output texel in the middle. In case of this algorithm, a Poisson disc kernel is used. The possion disc kernel has the nice property that it is easily re-sized and can therefore give different degrees of blurring, see Figure 12.

A Poisson disc is a disc of sample points that are put at random positions within the disc (not completely random, there should be a restriction on the minimal distance to any other point [Cook85]) . The coordinates are defined as 2D points from the origin. These coordinates can be scaled by an arbitrary radius and added as offsets to the current pixels texture coordinates to obtain the sample points.

Note that higher sampling densities give better results (since we get more complete information about the surrounding area), and that this density is reduced when the radius is increased. The result will not be as good with a larger disc unless the number of samples are increased (which is very expensive, more on these problems in section 3).

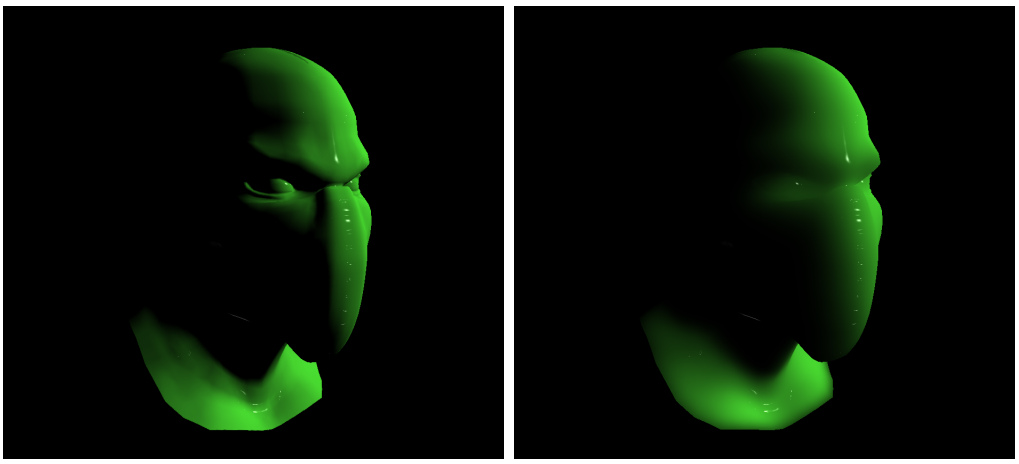
The Poisson disc is used to blur the diffuse lightmap and then as lighting for the inner layer(s). The amount of blurring depends on the distance between the layers, giving a greater scattering effect for deeper layers. This effect is illustrated in



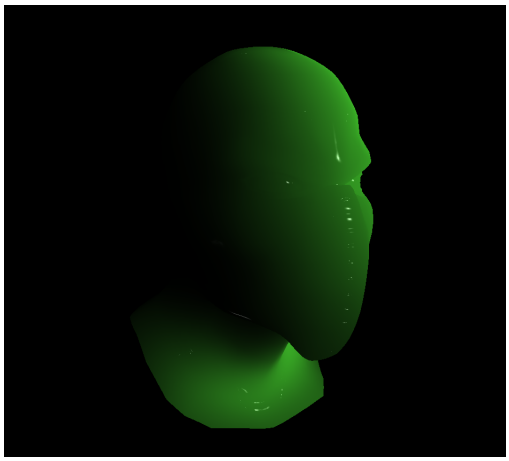
(a) (b)

Figure 12: Two Poisson discs of different sizes used to sample the surrounding pixels.

Figure 13. The ninja head in Figure 13(a) uses a lightmap without blurring. Figure 13(b) uses some blurring. Notice how the light is smoother and more spread out (particularly around the eyes). The last ninja head in Figure 13(c) has a considerably blurred light map, and gives a large spreading of the light. The sense of translucency is increased as the blur increases. The specular lighting is also important to give this translucent effect.



(a) (b)



(c)

Figure 13: Ninja head rendered with a diffuse light map and different amounts of blur as well as specular lighting.

The blurred lightmap takes care of the scattering of incoming lighting, but light also scatters on its way out of the material. To approximate this effect a blurring of the inner layers base texture is used. However, unlike the constant blurring of the lightmap which depended on the layer depth, the blurring of the base texture instead depends on the transmission distance within the material. This way, more blur is applied when looking deeper into the material.

The calculation of the transmission distance was described in section 2.2.1, equation 17. The transmission distance can be used straightforwardly as a scaling parameter to the Poisson disc sampling.

2.2.3 Putting it all together

All the above methods can now be combined into a coherent algorithm for a multilayer translucent material. The algorithm consists of the following steps:

- Transform the view vector V and the light vector L into tangent space.
- Sample all the outer layer textures to obtain the outer base colour, normal N and alpha value α.
- Compute the inner layers offset texture coordinate as well as the transmission distance.
- Render the model with diffuse lighting into an offscreen frame buffer with texture coordinates as vertex positions.
- Blur the light map based on the layer thickness.
- Blur the inner base map based on transmission distance.
- Calculate the NdotV value
- linearly interpolate between the layers based on alpha and NdotV[2]:
  $x = outerBase.rgb * outerDiffuse.rgb$
  $y = innerBase.rgb * innerDiffuse.rgb$
  $s = outerbase.a * NdotV$
  $x * (1 - s) + y * s$

[2]The outer diffuse light term can be calculated as usual or sampled from an non blurred lightmap. Note also that it is assumed that the alpha channel is stored in the outerBase texture.

3 Implementation details

This section describes various implementation solutions and challenges that cropped up during the thesis work. It also shows a few improvements to the basic techniques discussed in section 2. The next section will cover the subsurface texture mapping technique.

3.1 Subsurface Texture Mapping

This technique gave quite a few problems during the implementation. The following subsections will detail these problems and how they were overcome.

3.1.1 Fixing a bug

Equation 14 in section 2.1.3 showed the computation of the distance which the light travels for each layer to reach point M. This equation has been modified a slightly from the original in [Guillaume et al 2006]:

$$d_i \approx \min\left(||K'M|| - \sum_{j=0}^{i-1} \frac{D_j}{\cos(\theta)}, \frac{D_i}{\cos(\theta)}\right) \tag{20}$$

Combining this equation with equation 12 in section 2.1.3[3]:

$$L_{ri}(M, \omega_{in}) = L_i(K, \omega_{in}) e^{-\sigma_t^1 d_1} e^{-\sigma_t^2 d_2} e^{-\sigma_t^3 d_3}$$

and assuming that in the current calculation, $||K'M||$ only reaches into the second layer. It can be seen that the calculation of the latest distance d_3 will not be correct. The term $||K'M|| - \sum_{j=0}^{i-1} \frac{D_j}{\cos(\theta)}$ can be described in words as:
The total distance the light travels from the surface[4] to point M, minus the distance travelled in all previous layers (all except the current where M is located).

In case of the current example, the distance of the first layer would be subtracted from $||K'M||$, yielding the actual distance travelled in the second layer. Notice that this only makes sense when calculating distances that the light has actually reached. In this example layer three is never reached, but the formula still subtracts the distances travelled inside layer one and two from $||K'M||$. Since M lies inside the second layer $||K'M||$ is by definition smaller than the combined distance of layer one and two, resulting in a negative value for $||K'M|| - \sum_{j=0}^{i-1} \frac{D_j}{\cos(\theta)}$.

Equation 20 takes the smallest value of $||K'M|| - \sum_{j=0}^{i-1} \frac{D_j}{\cos(\theta)}$ and $\frac{D_i}{\cos(\theta)}$, which in this example is a negative value.

The negative distance is then used in equation 12:

$$L_{ri}(M, \omega_{in}) = L_i(K, \omega_{in}) e^{-\sigma_t^1 d_1} e^{-\sigma_t^2 d_2} e^{-\sigma_t^3 d_3}$$

Using a negative value in $e^{-\sigma_t^3 d_3}$ will create a very big value, which throws of the entire calculation. This is why a max function is used in equation 14. The max function will choose the value 0 instead of any negative value. This is also the correct value for these distances since the light never reach these layers.

[3] The equation assumes that exactly three layers are used, but can easily be extended by adding more $e^{-\sigma_t^i d_i}$ terms.

[4] Or more correctly the approximated surface

3.1.2 Noisy sampling

When the ray-marching algorithm samples the subsurface map, it does so at regular intervals. Samples of nearby pixels tend to be located at similar depths, which creates an appearance of layers of different shades (this is made clearer in section 4.1). These layers have quite distinct and non smooth edges and can be quite unappealing. Given a high enough sampling frequency these layers tend to blend together giving the apperance of a smooth three dimensional shapes, but this can be quite expensive. To make things look a bit better even at lower sampling frequencies, noise can be added to smooth out the distinct edges of these sampling layers.

The first idea for generating noise was to use the builtin *noise* functions of the OpengGL shading language. However it appears these have never gotten any real hardware support. An NVIDIA graphics card was used during the implementation of this technique and had, as it turned out, no implementation of the GLSL noise functions in it's drivers. According to the nvidia GLSL specifications, the noise functions always return 0 [NVIDIA 2006], and is therefore of little use.

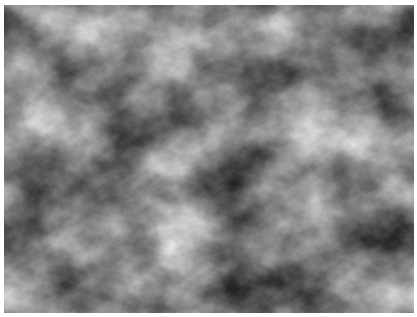


Figure 14: An example of a difference cloud texture.

Instead an easy approximation was choosen as a quick way of testing the noisy sampling technique. A difference cloud texture (Figure 14) was rendered in GIMP (The GNU Image Manipulation Program). This type of randomly generated data can be sampled in the pixel shader and used as an offset to the texture coordinates when doing the ray-marching. In order to get a sufficiently "random" offset, the difference cloud texture is first sampled once with the normal texture coordinates (this gives each point on the object a unique sampling as long as the texture coordinates are unique for each vertex). It is then sampled again with the value of the previous sampling. Doing this a number of times will give an offset that looks quite random.

Note also that the texture sampling gives values between 0.0 to 1.0, so in order to get an offset in all directions (in 2D), the values should be transformed into the 1.0 to -1.0 range. This is done simply by:

$$noiseOffset = noiseOffset * 2.0 - 1.0 \tag{21}$$

In order to make the amount of noise adaptable, a variable can be used to scale the noise offset. This variable can then be easily changed in real time. Figure 15 shows an example of real time control of several variables.

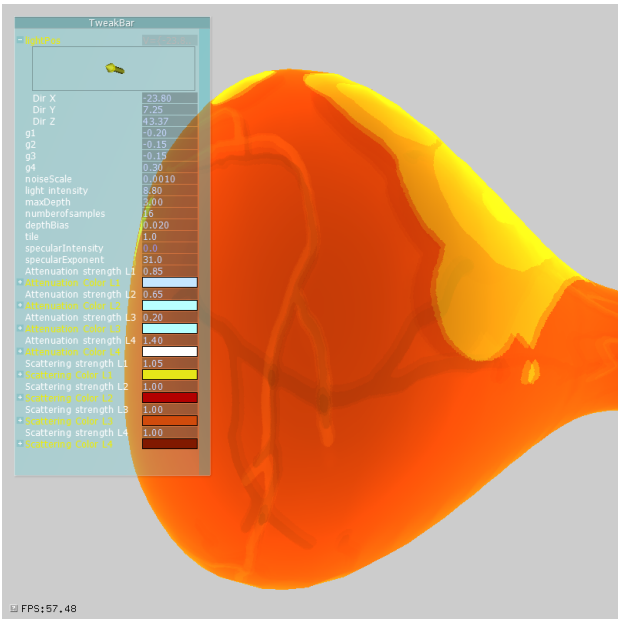


Figure 15: An example of real time changeable variables.

3.2 Multitexturing with depth parallax

The implementation of this technique was pretty straightforward. The one thing that was necessary to look closer at was the blurring of the light map. The reference article [Oat 2006] blurs the light map by using the Poisson disc kernel once. This type of setup tends to give good results when the Poisson discs radius is quite small but creates artifacts when the radius becomes to large. Instead of blurring, each sample contributes with a "copy" of the area surrounding that sample. Together all samples produce a blending of these copied patterns which surrounds the centre sample.

One solution to this problem is to do multipass blurring. The blurred image of the original lightmap is rendered into another off-screen frame buffer. This new texture is then used as input for the blur filter once again creating a blurrier image. This can be done for an arbitrary number of passes. The important thing to note here is that the Poission disc kernel radius can be set smaller than for a single pass and still produce the same amount of blur in the end. Samples stay closer together which in turn give a more correct blur effect.

Another way to make the light map blurrier is to render it into a texture with lower resolution. Since the image consists of less information, it will look blurrier when the graphics hardware samples the texture. Using interpolated values instead of the texels that would have been used in a higher resolution texture.

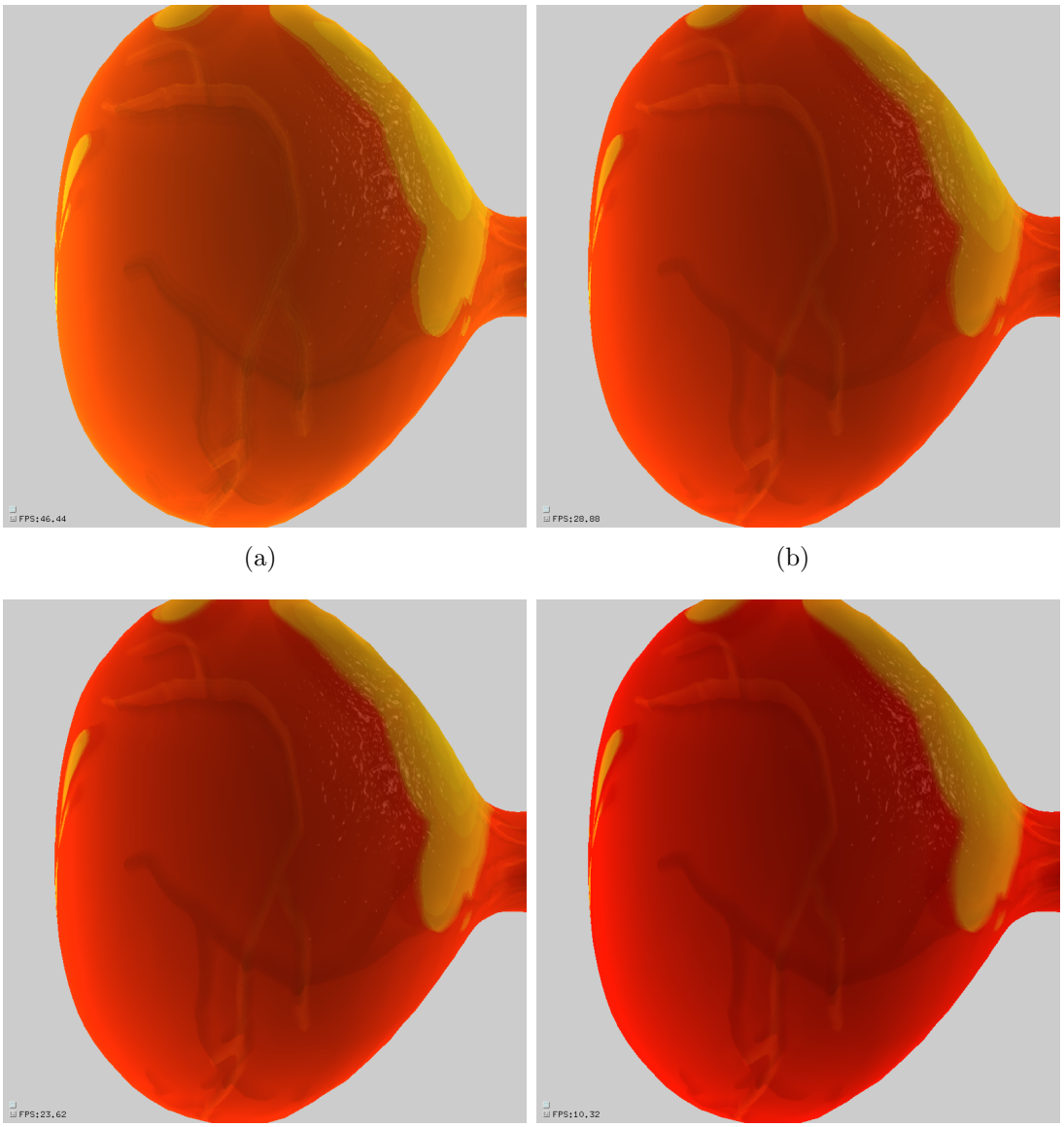
Both lower resultion textures and multipass rendering was used in the implementation.

4 Results

This section shows the result of the two implementations. Both techniques were implemented with C++/OpenGL and GLSL, mainly using a laptop with a intel core-2 dual-core CPU at 2.4Ghz and a Nvidia Geforce 8600M GT graphics card. Since one of the techniques are really computationally expensive, another computer was also used for testing, a 3Ghz quad-core intel core-2 CPU with an ATI Radeon HD4870X2 graphics card.

4.1 Subsurface Texture Mapping

This technique turned out to be very computationally expensive when the number of samples of the ray-marching increased. Renderings with different numbers of ray-marching samples are shown in Figure 16.



(a) (b)

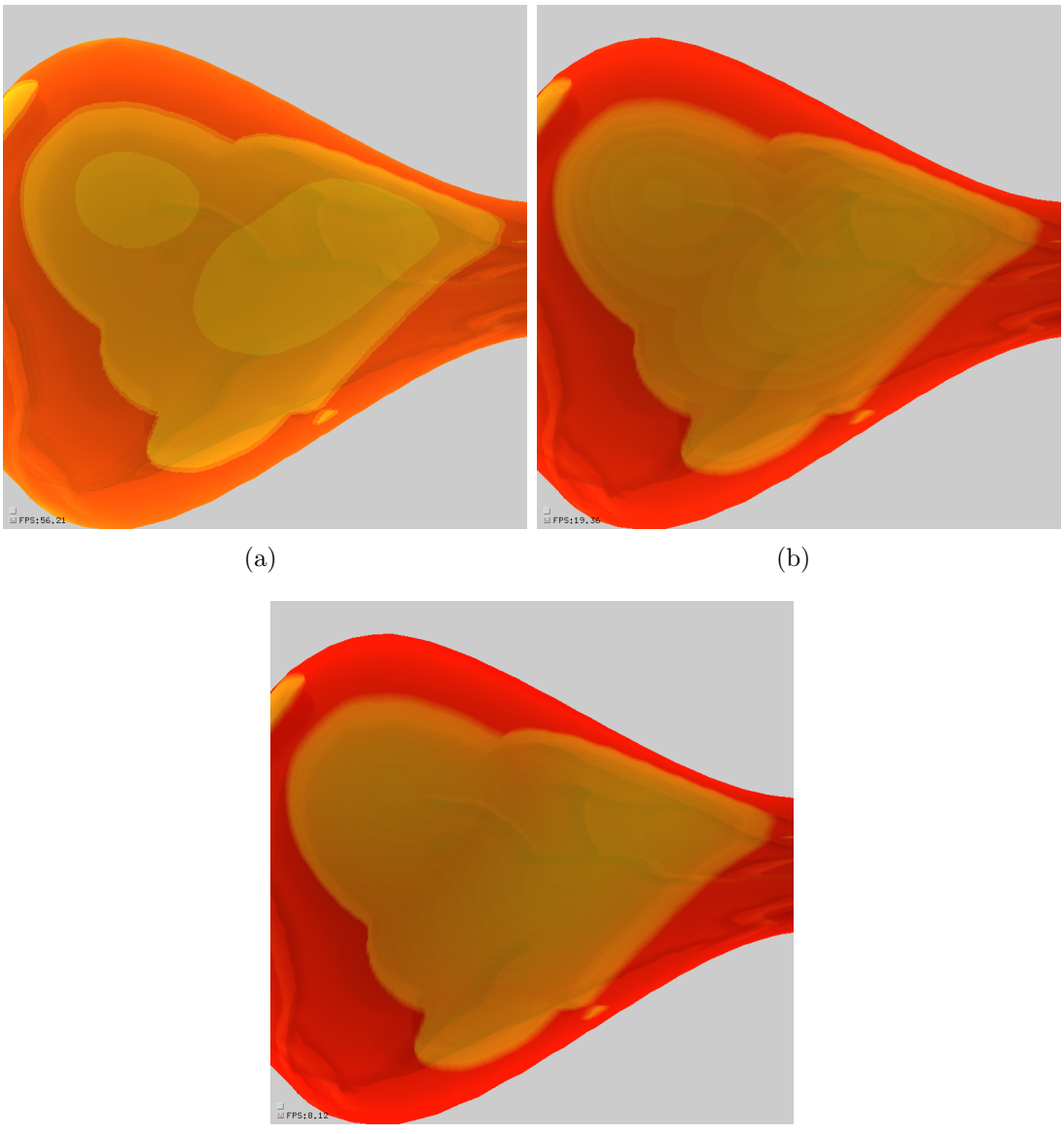
(c) (d)

Figure 16: Subsurface texture mapping with different amounts of sampling.

The image in Figure 16(a) used ray-marching with 15 samples, 16(b) used 25 samples, 16(c) used 35 samples and 16(d) used 100 samples. Notice how the details

in the images get more and more distinct and smooth as the number of samples increase.

The lines in the images are supposed to model veins and the yellow blobs model fat-tissue that form the upper layer. Figure 2(a) in section 2.1.2 shows the subsurface map being used in this rendering, although the first layer (the red channel in Figure 2(a)), has been clamped in the pixel shader so that it does not cover as large an area of the texture (making it easier to see the veins).



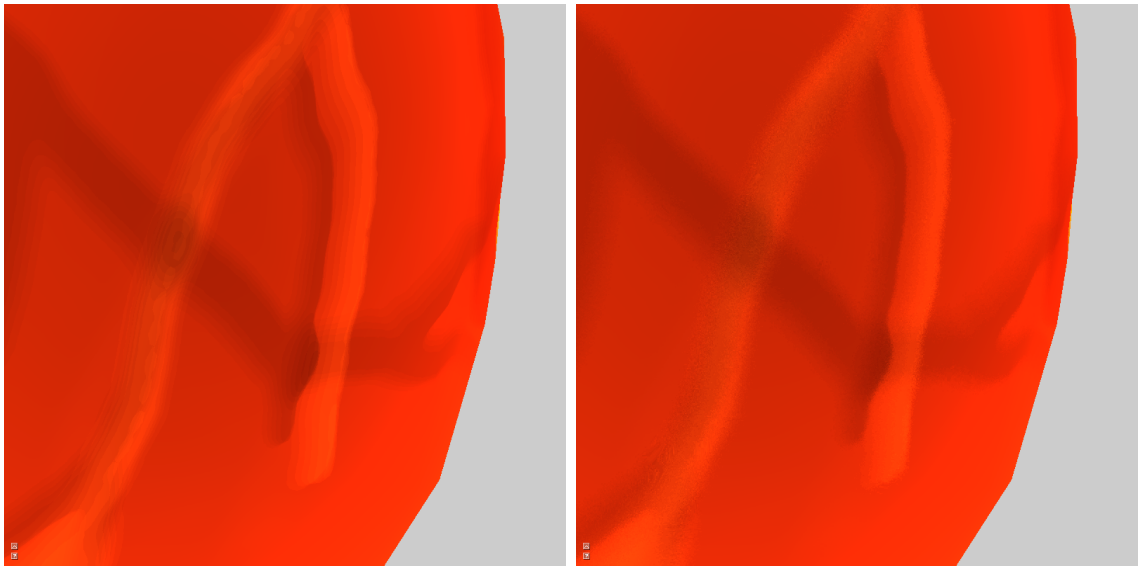
(a) (b)

(c)

Figure 17: The fat-tissue with diffent amount of sampling.

One thing to note about the 3D subsurface details are that they are built up by 2D layers placed at different depths. This is very apparent in Figure 17 where the first layers fat-tissue is shown with different amounts of sampling. For the rendering in Figure 17(a) the raymarching algorithm used 15 samples, 17(b) used 50 samples and 17(c) 100 samples. The layered appereance of the material is evident even in Figure 17(c). However, this apparent layering of the subsurface details are less apparent for details in the subsurface map where the gradient from dark to light changes rapidly, and not smoothly over a long distance of pixels. Compare the result

of the veins and the fat-tissue, the veins are modeled with a steep gradient change in the subsurface map, while the fat-tissue is not.
In order to alleviate the layered appearance, it may be a good idea to use noisy sampling. Although noise is introduced in the image, it may be preferable to the destinct edges of the layers. The noise can "blur" out the edges and create a smoother transition between the layers. This can be seen in Figure 18. The image in Figure 18(a) is rendered with 40 samples and clearly show how each sample contribute to a distinct 2D-layer. The rendering in Figure 18(b) on the other hand uses a randomised offset to the texture coordinates, resulting in noisy sampling. Notice how the edges that were easy to spot in Figure 18(a) are now quite hard to pick out. Although the noisy sampling does add a bit more calculation cost, it may actually improve performance since it may give comparably good results with low numbers of ray-marching samples. Removing the apearance of layers from the rendering without the noise usually requires a lot more samples and can be very computationally expensive.



(a) (b)

Figure 18: Rendering with and without noisy sampling.

4.2 Multi-texturing with depth parallax

Altough this technique is conceptually relatively simple, it can generate quite impressive results. The following results use the texture setup described in section 2.2, creating a kind of "skin-shader". A close-up photograph of a patch of skin is used as a basemap for the outer layer. A normal map and opacity map are generated from this image and a image of blod vessels are used as an inner layer base map (see Figure 8 section 2.2). The result of applying this shader to a ninja head model is shown in Figure 19. The light source in this picture is placed at the viewing position, resulting in a quite evenly lit model.

In the images in Figure 20 the light source have been move a bit to the side to give a clearer view of the shift from light to dark. In Figure 20(a) the blurring of the light map is turned off. This results in quite sharp transitions between the light and dark areas. The rendering in Figure 20(b) on the other hand have a more smoothed out transition between light and dark. Comparing some the areas around the eyes where there were small and very dark areas in Figure 20(a), there is more

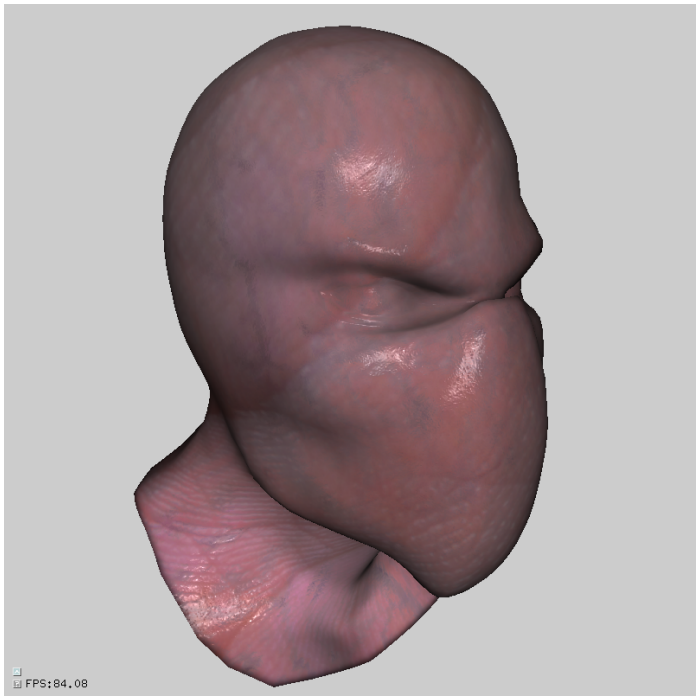
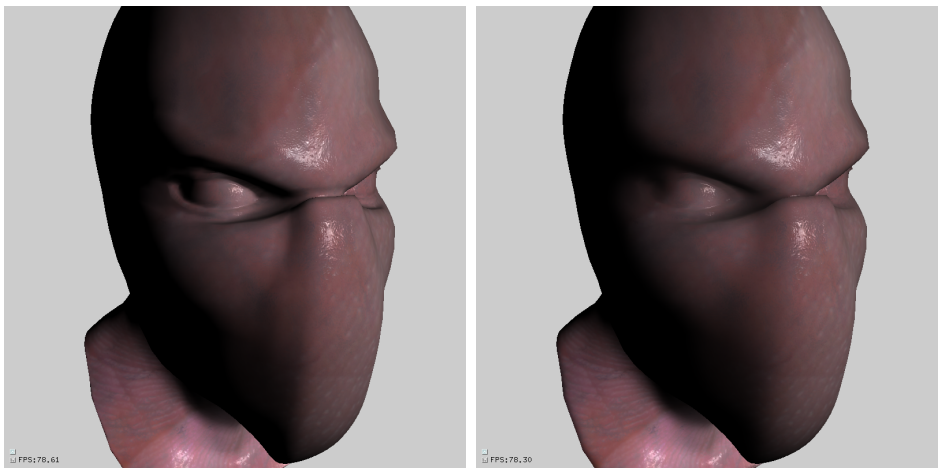


Figure 19: An example rendering with the material described in section 2.2.

smoothed out lighting in Figure 20(b). This makes it look like the light has entered the material, spread around and then exited the material from several places nearby.



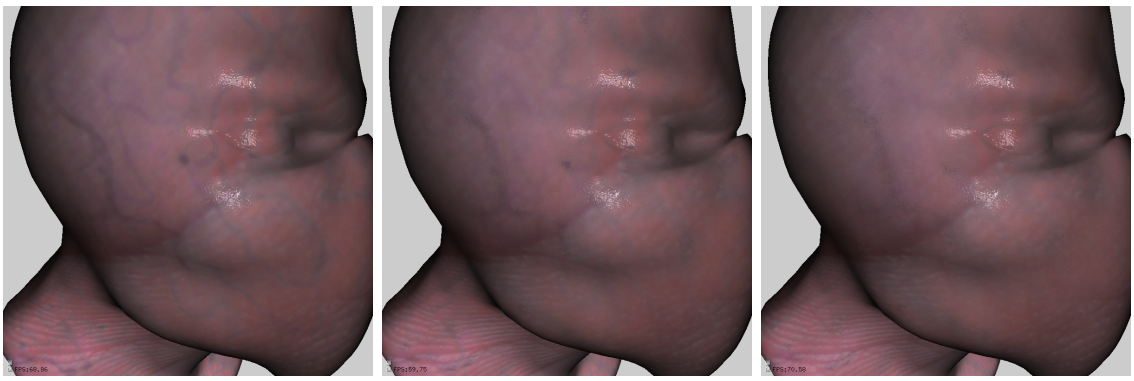
(a) (b)

Figure 20: Renderings with and without blurring of the light map texture. The area around the eyes are particularly interesting.

Another important visual element in the rendering is the appearance of the inner layer. When the inner layers depth is increased, the blurring of the inner layers base texture also increases (depending on the transmission distance in the material). This is however not the only thing happening. Since the transmission vector which decides the inner layers texture coordinates is calculated from normals sampled from the normal map, the normal pertubation can be different across the surface. This difference will give a refraction effect for the light coming from the inner layer. This can be observed in Figure 21. The difference in these images are the depth of the inner layer, increasing from left to right. Notice how the blood vessels become less

distinct as the depth increases.

The images also give a hint of the refraction effect due to the normal map. This refraction effect also makes it hard to see any distinct blurring of the inner layer (a normal map with less pronounced normal pertubation would give a clearer view of the blurring).



(a) (b) (c)

Figure 21: Renderings with different amounts of inner layer depth. The depth increases from left to right.

The depth parallax effect is quite hard to illustrate in images and requires motion to give it justice. It does however look quite convincing when moderate depths are used.

5 Conclusion

In this section different pros and cons of the techniques are discussed, as well as each techniques suitability in the application of rendering organic materials. Especially their suitability in the application of surgical simulation and the special conditions of such an environment.

5.1 Subsurface Texture Mapping

This technique can model the layers in a quite intricate way and is thus suitable for detailed subsurface rendering. However it comes with some major drawbacks, especially when considering the application of surgical simulation.

Since surgical simulation models interaction with soft tissues and organs within the body, it is very important that the geometry is flexible and can be pulled and prodded. This type of interaction requires deformable geometry. Because of the subsurface texture mapping techniques reliance on planar approximation, that need to be precalculated before rendering begins, it can only handle static geometry. This major drawback alone makes it quite unusable in surgical simulation, unless a method is devised to dynamically animate the planar approximations when the geometry is changed. Though this might be possible, it may require more effort than its worth, especially if arbitrary deformations are required.

The next big issue is that of performance. In order to achieve a good looking layered appearance, lots of samples are needed. This may be alleviated in the future as the graphics hardware improves, but as of right now, even a relatively

fast computer does not give any reasonable frame rates, not even at quite poor resolutions such as 800x600.

Another thing to note is that each layer is only given one scattering color and one attenuation color (the σ_t and σ_s parameters which are both RGB vectors). The color of the layer changes shade based on its deepness in the material and the attenuation color which controls how fast each color channel is attenuated. The result of this is that even though there is a lot of shading from lighter to darker, there is not a lot of variation in color. This leads to an almost cartoonish appearance of the material. A possible improvement for this problem is to blend each layer with some texture to get a more varied look. This would however make the technique even more computationally expensive.

Good results with this techniques requires some artistic talent when making the subsurface map. The results in this report had a pretty crude subsurface map and would benefit from more artistic and complex layers. The examples in [Guillaume et al 2006] show some more impressive examples of what can be accomplished.

Although this technique is very interesting theoretically, it is not very suitable in a practical implementation for surgical simulations.

5.2 Multi-texturing with depth parallax

This technique may be a lot simpler and less physically accurate, but it does give some pretty promising results. Nor does it suffer from the faults that made the other technique so unsuitable. No precalculation is needed and everything is computed within the vertex and fragment shaders. This means that it should be well equipped to handle deformable geometry. The technique is also quite fast and scales well with higher resolutions.

The result of this technique relies heavily on the quality of the texture maps. For accurate surgical simulation, the textures should be sampled from high quality photographs. The example presented in this report was quite crude in its texture composition, but with a bit more work and better photographic material, it should be possible to get a pretty accurate result.

One thing that needs to be taken under consideration with this technique is the model. The model needs to have unique texture coordinates for each vertex, otherwise there may be overlapping triangles when the light map i generated, which may result in triangles being rendered in the wrong order.

This technique should be quite applicable in a scenario where physical correctness is not as important. If only approximate visual appearance is needed, it should be quite useful for rendering in surgical simulations in particular as well as organic materials in general.

References

[Guillaume et al 2006] Guillaume François, Sumanta Pattanaik, Kadi Bouatouch, Gaspard Breton. "Subsurface Texture Mapping", INRIA Research report, July 2006

[Oat 2006] C. Oat. "Rendering Gooey Materials with Multiple Layers", ACM SIGGRAPH, Course 26: Advanced Real-Time Rendering in 3D Graphics and Games, 2006

[Policarpo et al. 2005] Fábio Policarpo, Manuel M. Oliveira, João L. D. Comba, "Real-Time Relief Mapping on Arbitrary Polygonal Surfaces", 2005.

[Walter et al. 2009] Bruce Walter, Shuang Zhao, Nicolas Holzschuch, Kavita Bala, "Single Scattering in Refractive Media with Triangle Mesh Boundaries", 2009.

[Cook85] Robert L. Cook, "Stochastic Sampling in Computer Graphics", 1985.

[Kolb et al.] H Kolb, E Fernandez, R Nelson, B W Jones, "The organization of the Retina and Visual Systems." http://webvision.med.utah.edu/index.html

[NVIDIA 2006] "Release Notes for NVIDIA OpenGL Shading Language Support", November 9, 2006.
http://developer.download.nvidia.com/opengl/glsl/glsl_release_notes.pdf

[Jensen 2001] Jensen, Henrik Wann, Stephen R. Marschner, Marc Levoy, and Pat Hanrahan. "A Practical Model for Subsurface Light Transport.", In Proceedings of SIGGRAPH 2001.

[Jensen and Buhler 2002] Jensen, Henrik Wann, and Juan Buhler. 2002. "A Rapid Hierarchical Rendering Technique for Translucent Materials." In Proceedings of SIGGRAPH 2002.

[Donner and Jensen 2005] Donner, Craig, and Henrik Wann Jensen. 2005. "Light Diffusion in Multi-Layered Translucent Materials.", In Proceedings of SIGGRAPH 2005

[Borshukov and Lewis 2003] Borshukov, George, and J. P. Lewis. 2003. "Realistic Human Face Rendering for The Matrix Reloaded." In ACM SIGGRAPH 2003 Sketches and Applications.

[Gosselin 2004] Gosselin, David. 2004. "Real-Time Skin Rendering." Presentation at Game Developers Conference 2004.

[d'Eon and Luebke 2007] Eugene d'Eon, David Luebke, "Advanced Techniques for Realistic Real-Time Skin Rendering", GPU Gems 3, 2007