Master in Computing
Llenguatges i Sistemes Informàtics

Master Thesis
– July 2011 –

Fast simulation of translucent objects

Student: *Christian Gascons Gomez*
Director: *Carlos Andújar*

UNIVERSITAT POLITÈCNICA DE CATALUNYA

Title:	Fast simulation of translucent objects
Author:	Christian Gascons Gomez
Advisor/Director:	Carlos Andújar
Date:	July 2011

Abstract:

Many common materials, including fruit, marble and skin, are somewhat translucent. Subsurface light transport models allow the simulation of translucent materials but at some rendering cost. On the other hand, there are some recent studies which analyze, from a human-perception point of view, which visual cues make an object look translucent or opaque.
Some factors, such as light source direction, color, contrast, blurriness and glossiness, have a strong influence on perceived translucency.
This project aims at developing new algorithms and empirical shading models for rendering objects that look translucent, with the emphasis on providing high-performance, perceptual-based shading rather than a physically-based solution. The idea is to devise new empirical shading models which produce physically-incorrect but visually equivalent renderings of translucent objects.

Keywords:	SSS Subsurface Scattering Real-Time Empirical Shader
Language:	English
Modality:	Research Work

Acknowledgements

I would like to thank my Master's Thesis advisor Carlos Andújar whose encouragement, guidance and support enabled me to develop this project. Without his help it wouldn't have been possible.

I would also like to express my gratitude to all my work colleagues, for supporting me day by day and for encouraging me to keep forward. In particular to Enric Sant, Ana Jiménez and Jordi Palomé.

I owe my deepest gratitude to my Family and Friends, especially Leandro Zabala who always inspired, encouraged and fully supported me along all the Master course.

Finally I am especially grateful to Marta, for all the strength that kept me keeping forward and for the hope she has given me during all this time.

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

Many materials such as milk, skin, marble and wax can be easily distinguished from others without any difficulty. This is because of the nature of their reflection properties. Objects made of translucent materials let light rays to get inside the object, travel inside the material and leave the object at a different point of the surface. This effect is called Subsurface Scattering.

The way it works is simple: the light strikes against a translucent object, scatters across it, and emerges at a different point of the surface, making it look smoother. When the light hits the surface of an object, the immediate reactions include reflection, absorption, refraction, diffusion and scattering among other effects. In real life, once a ray of light hits an object, the surface absorbs certain wavelengths of light based on its material, other wavelengths are reflected back and some others scatter across the object producing at the exit point, the color we see. The Fig.1.1 shows an example of this phenomenon.

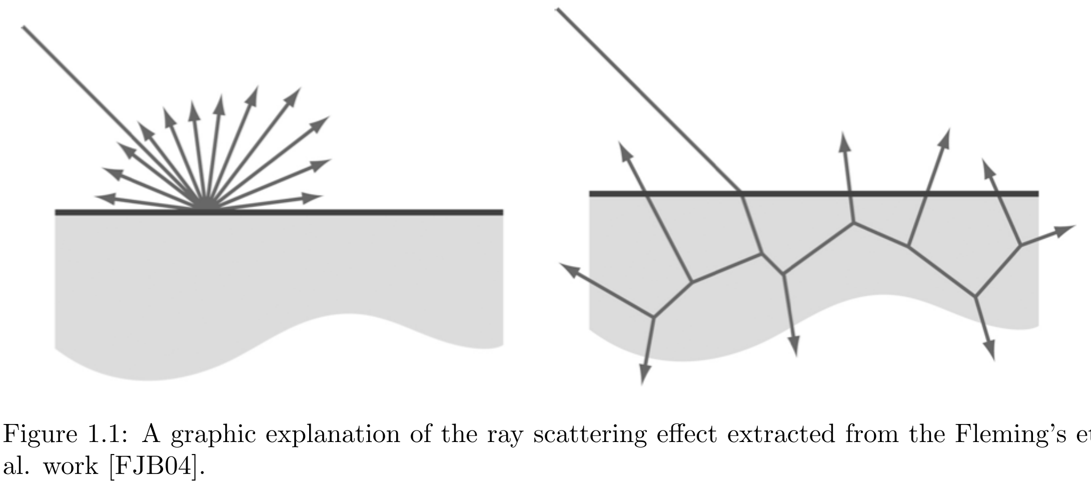


Figure 1.1: A graphic explanation of the ray scattering effect extracted from the Fleming's et al. work [FJB04].

Some recent studies have analyzed, from a human-perception point of view, which visual cues make an object look translucent or opaque. Some factors, such as light source direction, color, contrast, blurriness and glossiness, have a strong influence on perceived translucency.

Throughout the day we met easily with translucent objects: ceramic, wax, fruits, desktop erasers and even human skin is translucent under adequate light conditions. The light direction, for example, is an important factor for increasing the perception of an object translucency, and thus it has to be taken into account when trying to simulate Subsurface Scattering effects. The change of color hue and saturation, and the glossiness of the surface are also key factors influencing the perception of translucency.

Many of the translucent materials mentioned come together with effects of specular highlights across the object. It has been studied that, the more specular highlights has an object, the better translucent perception we will get from it. As the study done by Roland et al. [FB05] states, the human visual system ”expects” translucent materials to exhibit specular reflections.
An example of the importance of glossiness is shown in Figure 1.2, where the same model has been rendered without and with specular highlights. By adding specular reflection, the perception of the translucency increases as we are also able to see all the high frequency details of the model. Therefore glossiness of a surface is another key factor to take into account when simulating translucent materials.

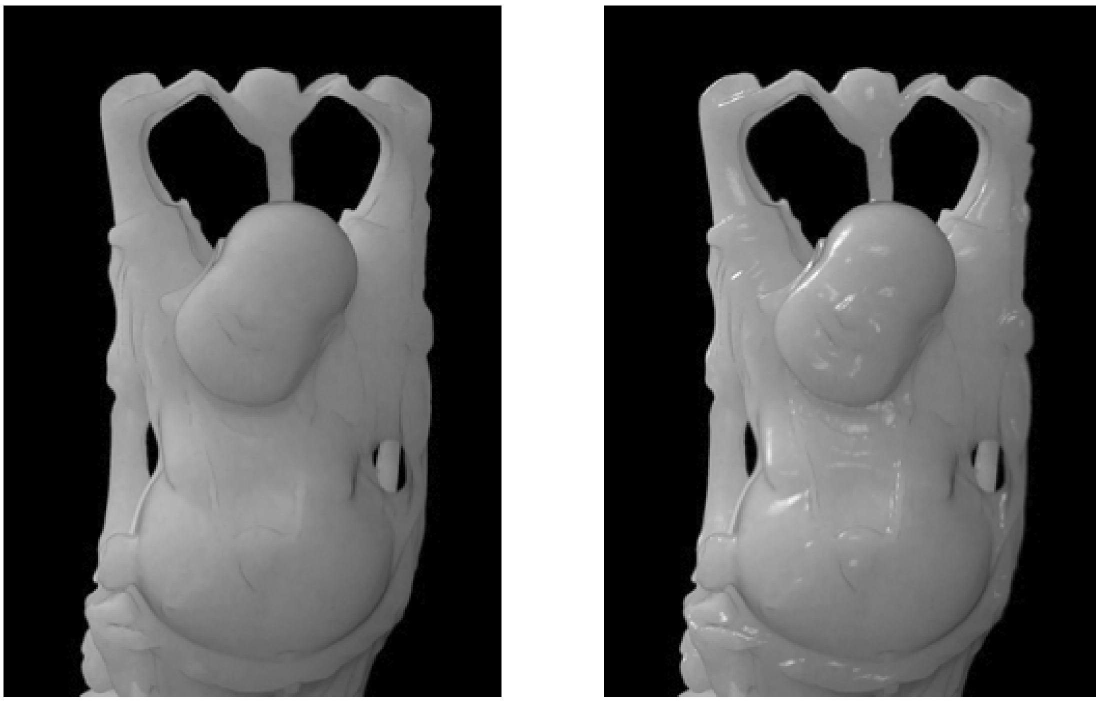


Figure 1.2: The Buddha model rendered without and with specular highlights. Images extracted from [FJB04].

The light passing through a translucent object will get out of the surface adopting a final color resulting from the mix of the light and the material colors, so to simulate Subsurface Scattering, a measure of the distance the light has traveled through the material must be obtained. Therefore, the more thick an object is, the less backlight will emerge on the surface. As we shall see, merging the color of the incident light with the color of the material is absolutely necessary for increasing the translucency appearance. We have also observed that using saturated colors for the object's albedo allows the human visual system to better perceive translucency effects, in particular in thin parts of the model. The image in Fig.1.3 shows an example of this.

We also observed that the weight of the diffuse reflection of direct incident light also influences the perception of translucency. Decreasing the contribution of the diffuse term in the final color and increasing specular highlights leads to images with much more translucent appearance.

Many techniques have been proposed in recent years achieving plausible results. Some of them made use of pre-processing techniques which did not allow them to work at interactive rates. Our goal is to simulate subsurface scattering in real time by developing new algorithms and empirical shading models for rendering objects that look translucent. We focus on empirical techniques that could make the human visual system to perceive translucency in opaque objects with low rendering cost.

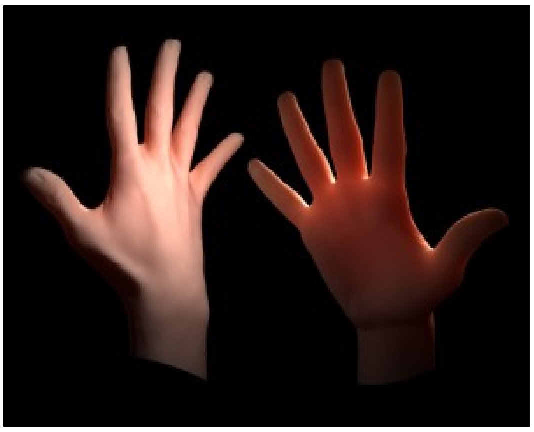


Figure 1.3: An example of Subsurface Scattering. A hand with backlight illumination rendered from two different points of view. When the object rendered is between the observer and the light source, the thin parts of the 3D models increase the translucency of their materials substantially.

Some key ingredients of our approach include the use of pre-computed per-vertex ambient occlusion, the computation of the object's thickness in screen-space through depth maps, and the use of Gaussian blur to smooth thickness before it is used in the proposed lighting equation.

The rest of this document is organized as follows.

Chapter 2 contains the state of the Art on the different techniques in the literature for making an opaque object look translucent. Section 2.1 drives us through the previous techniques studying which parameters affect the perception of the translucency; Section 2.2 summarizes a few techniques for achieving translucent effects. Chapter 3 explains in detail our algorithm and the key points that lead us to the proposed solution. The most important part of the algorithm, which is the Lighting Equation, is specified in Chap. 3.1.1. Chapter 4 presents the results of our implementation. We will find figures, tables detailing the performance and surface analysis for the depth maps extracted of the algorithm as well as a user study conducted to analyze how the variation of some parameters affected the final perception of the object.

Finally, we present concluding remarks in Chapter 5.

The list of appendices available in this Master's Thesis provides the reader with the necessary knowledge for understanding the features of the application that has been implemented. In addition, a global review of the User interface and the full code for the shaders to be executed in the graphics card is provided.

CHAPTER 2

Previous Work

2.1 Perception of translucent materials

Fleming et al. [FJB04] discuss some physical factors influencing the perception of translucent objects, such as the fact that light-source direction can alter the apparent translucency of an object, finding that objects are perceived to be more translucent when illuminated from behind.

The absorption and scattering coefficients specify the probability that a photon will be absorbed or scattered when traveling a given distance within the material. Koendering and van Doorn [2001] pointed out that translucent materials have an appearance that differs from the traditional Lambertian materials.

Highlights occur when light is specularly reflected from the surface of an object. The human visual system may "expect" translucent materials to show specular reflections. The authors found that highlights can contribute to the visual impression of translucency. They point out too that the glossy surfaces look more translucent than the surfaces without highlights. For the sake of clarity we have divided the author's main findings in [FJB04] in two sections: Image measurements and their conclusions.

Image measurements They study and an identifiy the process regarding the image cues that carry information about translucency. The authors measured how various image properties such as color saturation and image contrast vary with parameters of the BSSRDF.

The authors state that when light passes through a colored translucent object, it is progressively filtered, and emerges colored. Hue, saturation and intensity can all vary as a function of the distance traveled by a ray through a translucent material. The authors found that color saturation can affect the way a translucent object appears to 'glow'. Although saturation variations can affect the perceived translucency, they are insufficient on their own to yield an impression of translucency.

Fleming et al. also studied the image contrast effect. They state that, when illuminated, translucent objects become 'filled' with light. Points on the surface that do not receive any direct illumination can nevertheless receive light from within the body of the object. Conversely, regions that receive strong direct illumination tend to dissipate the incident light by transmitting it to other parts of the object. In addition, the authors specify that the visual system can perceive a translucent object appear opaque simply by changing the intensity histogram. So although the intensity histogram captures something important about translucent objects the authors emphasize that is insufficient alone to yield an impression of translucency.

Perceived translucency does change when the light source moves. So it is not solely the proportion of light and dark in an image, but their spatial relationships that makes an object look translucent. Blur plays a key role in the distinctive soft appearance of translucent materials, but again, is insufficient to produce a percept of translucency. If the intensity distribution is held roughly constant, adding blur has a small effect on perceived translucency.

Conclusions The authors extract some clear points about the perception of translucent objects.

- Translucent objects look most translucent when they are glossy and lit from behind.
- If we want translucent objects to look glowing and warm, color saturation should be positively correlated with intensity.
- If we want them to look icy or dilute, the correlation should be negative.
- Translucent objects should be lower contrast than opaque ones. Sharp cast shadows should be avoided as they make objects appear hard an opaque.
- Blur and loss of detail gives translucent objects their characteristic soft appearance.
- An opaque object can not appear translucent simply by blurring out the details.

2.2 Rendering of translucent materials

The existing literature provides a wide variety of previous work regarding Subsurface Scattering effects. These include pre-processed algorithms, and real-time techniques. On the next lines some of these techniques will be discussed and analyzed in order to know those that led us to the algorithm that we present.

2.2.1 Real time techniques for simulating translucent objects

Under these lines we will review some techniques from many different authors that achieved very good results by applying their real-time algorithms.

Rendering of local subsurface scattering

Mertens et al. in [MKB$^+$03] introduce a rendering algorithm which operates in image-space. They demonstrate the applicability of their technique to the problem of skin rendering, for which the subsurface transport of light tipically remains local. The implementation shows that plausible images can be rendered interactively using hardware acceleration. A Hierarchical integration technique was proposed in [JB05] in order to use an analytical model for subsurface scattering, unfortunately, this technique does not appear to allow interactive rendering, which is what the authors of this paper seek. Their goal is to render deformable translucent objects at interactive rates under varying lighting and viewing conditions. One approach was done by Hao et al. at [HBV03] where they achieved to render translucent but rigid objects in real-time.

Real-time rendering with complex surface BRDF's has been widely studied. Unfortunately, these techniques cannot handle subsurface scattering because they assume that light is directly reflected and not scattered inside the object.

Mertens technique renders an irradiance map from the eye view, while importance sampling acts as a filter. The steps that the authors followed have been explained as follows:

Subsurface reflection equation The author's algorithm use a BSSRDF, which is an eight-dimensional function, expressing what fraction of differential light energy entering the object at a location x_i from a direction w_i leaves the object at a second location x_0 into direction w_0. This function depends on the object's geometry.

Thus, the authors state that subsurface scattering can be modeled adequately using two components: *single scattering* and *multiple scattering.*

- *Single scattering* accounts for light that is scattered only once inside the medium.
- *Multiple scattering* diffuses the incident illumination, such there is almost no dependence on the incident and outgoing direction anymore.

They finally get the following subsurface scattering reflectance function:

$$S(x_i, \omega_i; x_0, \omega_0) = \frac{1}{\pi} F_t(\eta, \omega_0) R_d(x_i, x_0) F_t(\eta, \omega_i) \tag{2.1}$$

Dipole Source approximation They need to choose a model for the BSSRDF and a model for R_d which is the four-dimensional function for calculating the multiple scattering. This model has to be adaptable to different materials and should allow for importance sampling to speed up the computation later on.

Their elegant solution works as follows:

An incoming ray at a given position on a homogeneous, planar, and infinitely large and thick medium is converted into a dipole source. One of the sources of the dipole is placed at a distance z_v over the surface, and the other one at a distance z_r, below the surface, at x_i. Then, R_d is obtained by summing a sort of illumination contribution at the starting point due to the two sources near x_i. Finally, the result is a function of the distance r between x_i and x_0.

Importance sampling of the BSSRDF Once working on the solution they presented, the authors derived an exact importance sampling[1] scheme for the BSSRDF model previously presented by Jensen et al. They had to integrate over the whole surface of an object and sum up all the contributions due to subsurface scattering. As an efficient integration to be performed they used the importance sampling because their goal is to find sample distances which are distributed according to R_d.

[1]An importance sampling is a general technique for estimating properties of a particular distribution, while only having samples generated from a different distribution rather than the distribution of interest.

Integration over the surface. They found two problems in this context. Firstly for a given importance sampling distance at a point on the surface, it is not trivial to construct a location at this distance directly on the surface. This would require access to geometry, as well as a complex search routine. So as a solution for this problem the authors preferred to construct the samples in the tangent plane of that point.

The second problem is the acquisition of irradiance information over the surface. To solve this issue, they rendered the irradiance once for one reference views into a 2D image or texture map.

Summarizing, for each point to shade, they construct a set of samples in the tangent plane as the Fig. 2.1 shows, and then project that point on the surface. For each sample point they look up the irradiance in the irradiance texture.

Their implementation On their implementation they found some artifacts caused by the variance, which are inherent to Monte Carlo techniques. The measures that the authors took to alleviate this were: Stratified Sampling and Combined Sampling[2].

Their results The authors of the paper achieved rendering speed from 4 to 5 frames per second for a 500x500 image on an ATI Radeon 9700Pro. They compared their results with Jensen's [JB05] and found small differences at sharp boundaries because no contributions were gathered from some hidden parts of the model as those were not visible in the irradiance image. The Fig. 2.2 show us a sample of the authors work. The whole procedure is executed entirely on the GPU, and no pre-computation was required on the host CPU, except for the marginal cost of generating importance samples and the R_d texture.

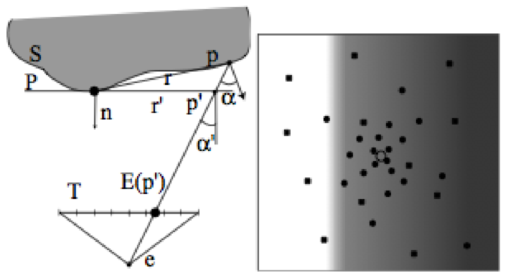


Figure 2.1: On the left image the geometry for sampling the irradiance in the tangent plane. The irradiance at the projected sample point p is retrieved in the irradiance texture T. The right image shows the combined sampling of importance samples.

Real-time subsurface scattering on the GPU

The authors of this publication [BBNM07] evaluate both single scattering[3] and multiple scattering by using piecewise linear and ring-based approximations of the surface in the fragment shader. They get rid of pre-computations by computing light transport on the fly. Before each frame, they compute a depth map from the light point of view recording the depths of points on the lit surface.

Subsurface scattering: They compute it with Monte Carlo's technique on the GPU, but it requires too many texture accesses on low-end GPUs. To accelerate the process they rely

[2]The authors generate a set of uniform samples together with the importance sample set.

[3]They take exponentially spaced samples inside the volume along the view ray and compute the lengths of the exit rays towards the light.

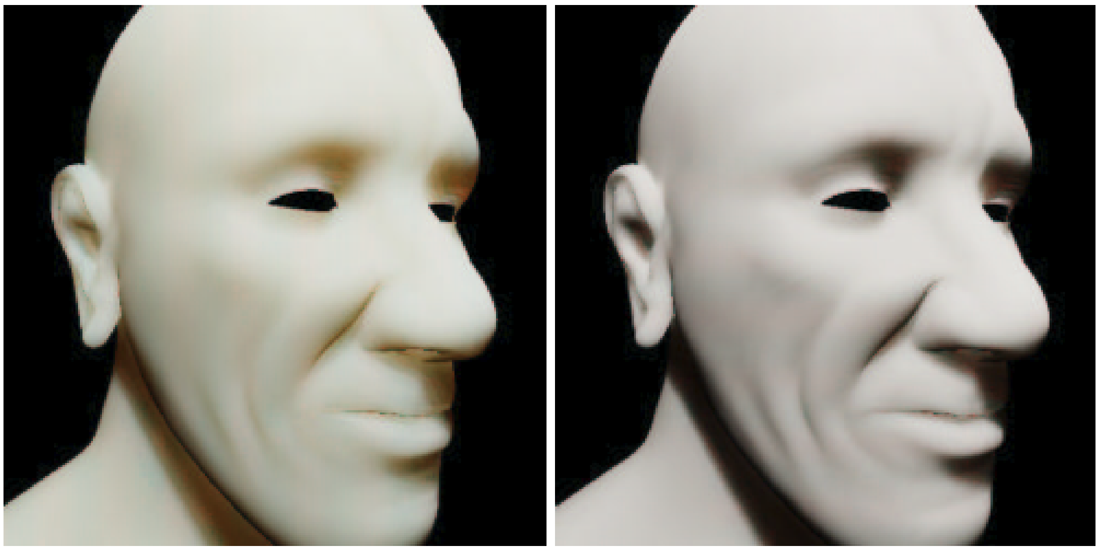


Figure 2.2: A sample of the results that the authors generated with their algorithm in [MKB+03]. In this model, milk and marble materials were applied to their models.

on the fact that the subsurface light transport function is constant for all the points on a ring around a given point and the use of Variance Shadow Maps [DL06].

Their implementation though, has a main limitation, which is quality as it can be seen in the Fig. 2.3. Because using depth maps from the light's point of view sometimes can cause aliasing artifacts and Mach Bands. As future work, the authors commented that using multilayered materials and textures would definitely improve realism.

Real-Time Realistic Skin Translucency

Jimenez et al. in [JG10] state that the the simulation of Sub-Surface Scattering is challenging because it must correctly simulate the light transport and capture its appearance. Human skin has multiple translucent layers which makes it even more interesting. Anyway, there is a big drawback, which is that the errors will be more noticeable in skin than in for instance, in wax.

Subsurface Scattering (SSS) is usually described in terms of Bidirectional Scattering Surface Reflectance Distribution Function (BSSRDF). Jensen et. al. provided a huge step and made SSS practical. Afterwards, the movie industry adopted most of their techniques and many authors published their own techniques; in order to achieve better results they used depth maps to estimate distance that light travels inside an object, blurred 2D diffuse irradiance textures and used many other techniques. In recent work Jimenez et. al. solved the BSSRDF gathering information like accumulating the Gaussian convolutions on the fly. After them, Ralf Habel et. al. represented SSS in real time pre-computing the image convolution required. Let us go step by step and get some background.

A *Convolution* is a mathematical operation applied on two functions, producing a third function viewed as a modified version of one of the original functions. The origin of these convolution images comes from the diffusion profile. For multilayered material such as human skin the light: Enters in the object, interacts with each of its layers and exits at various points

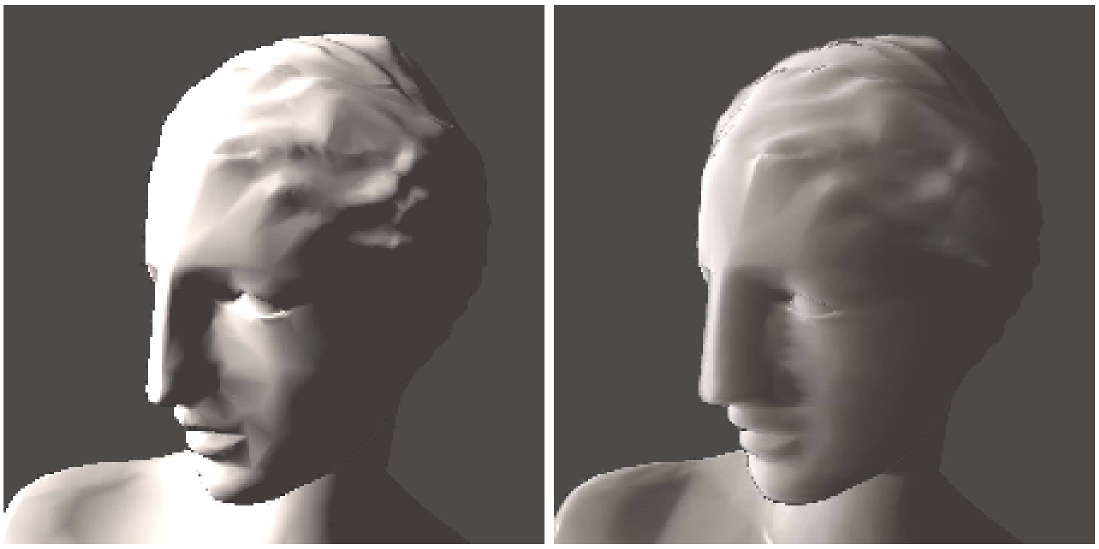


Figure 2.3: The image show a marble statue without (left) and with (right) subsurface scattering algorithm applied. Many artifacts and Mach bands are visible as the authors say because of the use of shadow maps. This image is extracted from the publication [BBNM07].

around the incident point. Each of those layers absorb and scatters the light differently. It can be defined as a function of distance r to the incident point R(r). To obtain such diffusion profiles they rely on diffusion theory and arrive at the classic diffusion equation.

Consider a point P(x,y) on the surface. The goal is to obtain the contribution of all the points around P. Part of the light arriving at the adjacent points will penetrate into the object and exit at P. The diffusion profile can be written as a 2D convolution, although these are costly for real-time but they can be separated into faster 1D convolutions.

Their technique Some authors presented a technique using irradiance maps, which is a point cloud on where any point in 3D space represents the light arriving at this point from all possible directions. This presents some advantages and some drawbacks. As an important advantage, realistic images can be achieved. As a drawback: there is not an available technique in screen space.

The authors wanted to translate the simulation of scattering effects from texture space to screen space. This has some consequences:

- Less information to work with.
- Loss of irradiance in all points of the surface not seen from the camera. (Only visible pixels are rendered)
- Not possible to calculate the transmittance of light through thin parts.

The authors finally built an heuristic that let them approximately reconstruct the irradiance on the back of an object producing images whose quality is on a par with photon mapping (offline render).

Their algorithm builds on two approaches, the first one: Green's approach, who's technique uses depth maps to estimate the distance that a light ray travels inside an object rendering the scene from the light's point of view and creating a depth map storing the distance. But this has a drawback: it only works with convex objects. Their second approach is Eugene d'Eons, which is texture-space based. His technique relies on calculating the irradiance at each point on the surface. As a solution, Eugene d'Eons used translucent shadow maps store depth z, irradiance, and the normal of each point on the surface nearest the light.

Finally, the authors building on the previous ideas presented a physically based transmittance shader. They need a function $T(s)$ that relates the attenuation of the light with the distance traveled inside an object.

Remarks For a great range of thin objects, they approximate the normal at the back of the current pixel normal when the back and the front surfaces are parallel. So they assume they can replace the exact normal N_a at a point A with the reversed normal N_c of the current point C. as the Fig. 2.4 shows.

When looking at a backlit object from the front the viewer does not have accurate info of the irradiance of the back so again, they assume that they can predict the irradiance at the back using some heuristic.

For materials with thick surfaces transmittance is a very-low-frequency phenomenon as the light is diffused as it travels inside an object. The albedo is a measure of how strongly it reflects the light from light sources such as the sun and it is therefore a more specific form of the term reflectivity. In human skin, albedo does not vary dramatically over its surface, it maintains a similar tone. As their last assumption, they safely use the albedo c at the front to approximate irradiance at the back of the object.

Their algorithm Given a diffusion profile $R(r)$, the transmitted radiant exitance $M(x, y)$ is the convolution of the incoming irradiance with the diffusion profile.

Rendering For rendering they need to add the contributions of reflectance and transmittance. Reflected and transmitted lighting can not happen simultaneously avoiding double contribution so, instead of blurring the distance traveled inside the object they store the transmittance and reflectance together, using screen-space Gaussian convolutions to blur them. Finally, a texture $T(s)$ encodes the transmitted lighting as a function of distance.

The authors though, used shadow maps for depth approximations, which comes with artifacts that can appear around the projection edges because the objects of the background are projected onto the objects edges. As a solution the authors shrink the object in the normal direction, while querying the depth map.

Thus, they get great realism to the images, but the transmittance algorithm does not require irradiance textures, it requires standard shadow maps, which they use for storing the distance from the objects nearest to the light and they create them by rendering the scene from the light's point of view which differs on our technique as we generate the depth maps by rendering the scene from the observer point of view. The authors too, make a different use of the shadow maps as they do not use them for checking if a pixel must be shadowed. Instead they use it only for obtaining the distance the light traveled inside the object. Their translucent shadow

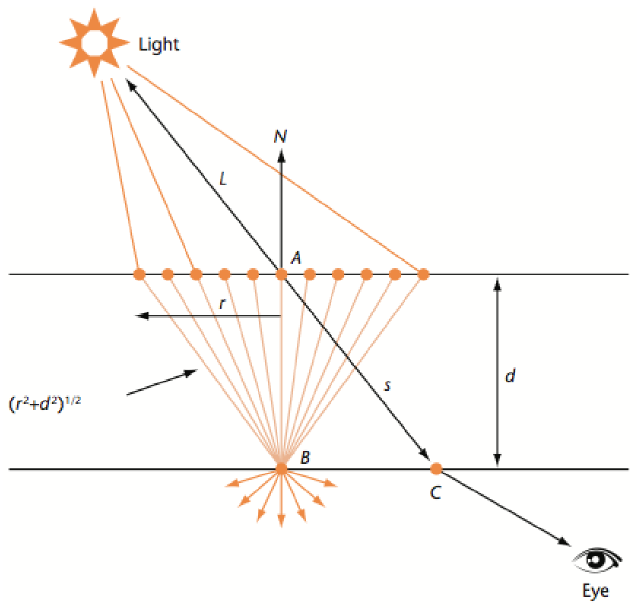


Figure 2.4: The image extracted from Jimenez et al. [JG10] shows that the radiant exitance at the point C is approximated by the radiant exitance at the point B because is faster to calculate using the irradiance information E around point A.

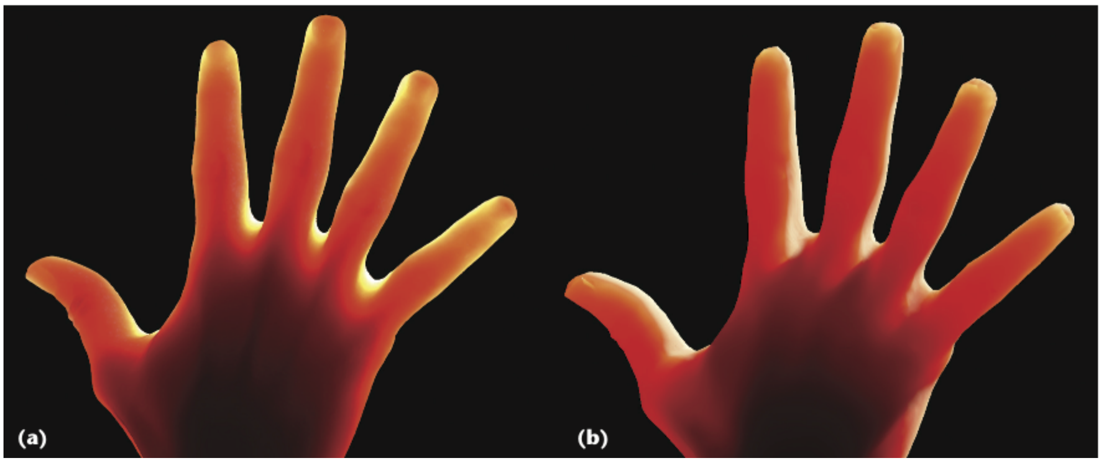


Figure 2.5: This figure shows the comparison of the photon mapping method and their algorithm. The light is only influenced by the epidermis and results in a yellowish tone. The photon mapping technique required 15 minutes to render whilst their algorithm only required 5 milliseconds per frame.

maps store depth z, irradiance, and the normal of each point on the surface nearest to the light, whilst in our technique we only use them for computing the *thickness* of the model.

They too store the transmittance and the reflectance together instead of blurring the distance traveled inside the object, which would be the thickness in our algorithm. Afterwards they use the screen-space Gaussian convolutions to blur them, as well as our technique, which generates good results. It also requires too less memory storage than previous approaches because it only requires the depth z eliminating the need to store the (u,v) coordinates for the shadow map. Their technique only accesses memory twice per pixel in order to get the values irradiance and depth values.

2.2.2 Non-interactive rate techniques

Pharr and Hanrahan [HK93] introduced the concept of scattering equations, Jensen et al. [JSM+01] proposed modeling the scattering of light in translucent materials as a diffusion process. The author's technique only works well in highly scattered media though, where Monte Carlo ray tracing becomes very expensive [Sta95]. Then, Jensen et al. suggested a simple analytical dipole diffusion approximation and found this model to be in good agreement with measurements of light scattered from translucent materials. They used this to formulate a complete BSSRDF (Bidirectional Scattering Surface reflectance distribution function) [NRH+92] sampling the incident fix on the surface. The BSSRDF (Jensen et al. [JSM+01]) is much faster than Monte Carlo photon tracing. But it is more expensive to evaluate than a traditional BRDF since it requires sampling the incident flux distribution at the surface and is particularly expensive for high translucent materials. Only includes internal scattering in the material due to direct illumination from the light sources.

The approach of the authors of the publication [JB05] (Fig. 2.6) is based on **two key ideas**:

1. Decouple: Computing irradiance at the selected points of the surface and evaluating a diffusion approximation using pre-computed irradiance samples.
2. Rapid hierarchical evaluation of the diffusion approximation using pre-computed irradiance samples. (Faster than (1) because it only evaluates the incident illumination once at a given surface location). That is efficient for highly translucent materials where sampling the BSSRDF is costly.

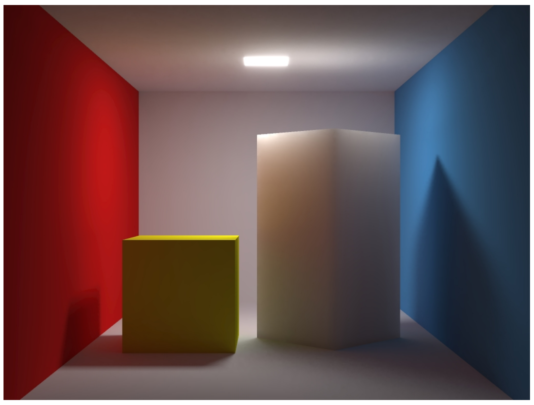


Figure 2.6: A global illumination scene with a translucent box generated in the publication [JB05].

Among their features, which deserve to be discussed, comprise: the possibility of computing the effects on indirect illumination on translucent materials, the fact that their hierarchical evaluation is deterministic and the Intrinsic scattering properties that can be computed from 2 parameters: Diffuse reflectance and the average scattering distance.

CHAPTER 3

Our Approach: Fake Translucent Objects

3.1 Our approach

Our goal is to achieve fake translucent effects on 3D models by using empirical techniques. At the beginning of the project that idea led us to perform an accurate study of the different existing techniques.

Many authors coincide on some relevant factors such as the glossiness of the object, its specular highlights or the light position where the object is lit from, for that reason our very first step on this project was to know exactly how the perception of the translucent materials affect the human visual system. We implemented a main application to run our tests on and load 3D models on Wavefront formats. As soon as it was implemented, we started working on the shader engine.

Our aim is to find the key aspects that make an object look translucent. Among all the possible options we decided to stick out some of them such as the thin parts of a model, which will receive rays of light as well as the thicker parts but exposing a different behavior on the exiting points of the surface (when the rays of light emerge colored). We will achieve the fact of highlighting the thinner parts of the model by calculating the thickness of the model. Note that this technique has been implemented in order to make objects look very translucent.

We also want to highlight the glossiness of the objects, which as previous studies state, it has a great importance. We manage to increase such factor by implementing the *Blinn-Phong* technique on our shaders and adding to the algorithm a multiplier for increasing the specular highlights of the model. As the previous studies specify, the perception of a translucent object is higher when it is lit from behind. Thus, the reader must take into account that this implementation will assume that there is always an artificial backlight located behind the object. For such reason, the very thin parts of the models will look brighter. Anyway, a light has been placed on scene and will show the specular highlights on the model as we move it across the scene.

It has been proved too that the smoothness of the model increases the perception of translucency, hence we wanted to slightly smooth the model without modifying the final appearance of it. In order to achieve such effect, we did not apply a smoothing technique directly on the model, but on a texture containing the depth values of the model's second layer faces from the observer point of view. We state that the diffuse contribution of the model somehow affects on the perception, and it has been proven that decreasing it, directly increases the translucent perception.

Thus, we need to get the model's thickness to highlight this parts of the model. Many papers state depth maps have a considerably perception of translucency, the Fig. 3.1 is an example of this. As a first idea we rendered the model, getting the depth map of the model's back faces.

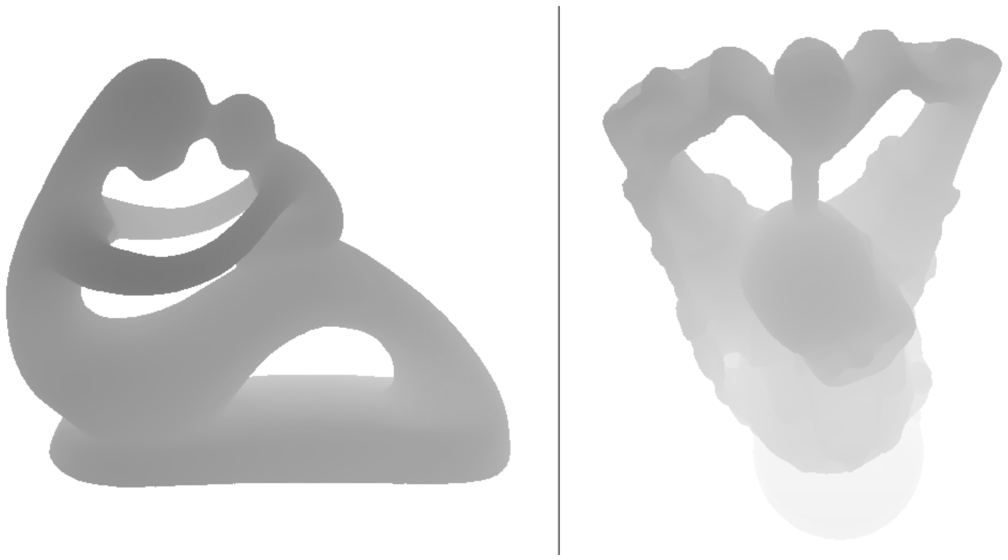


Figure 3.1: The depth map generated by rendering two different models, fertility and the Stanford's Happy Buddha.

We are only interested on computing the thickness in screen-space, i.e. the thickness of the visible parts of the model. Hence, we need to get the linear depth from the OpenGL depth buffer and visualize it. Depth values in screen space are non-linear, therefore we had to transform the z depth value from normalized device coordinates to eye-space coordinates in the fragment shader (after the perspective projection, the depth values are non-linear, for such reason there is more precision close to the camera and less precision far from the camera).

Thus, the process of transforming the depth values works as follows: We first get the depth at the back layer (which is subjected to a Gaussian Blur to be detailed below), and also store the fragment's depth as Eq. 3.1 specifies.

$$\begin{aligned} zBack &= smoothDepthXL(backDepth); \\ zFront &= gl\_FragCoord.z; \end{aligned} \tag{3.1}$$

Eq. 3.2 shows the how we get the clipping planes (zNear and zFar) values.

$$\begin{aligned}
&A = gl\_ProjectionMatrix[2].z;\\
&B = gl\_ProjectionMatrix[3].z;\\
&zNear = -B/(1.0 - A);\\
&zFar = B/(1.0 + A);
\end{aligned} \tag{3.2}$$

Finally, in Eq. 3.3 we normalize the depths at the back and front faces and transform them to eye-space.

$$\begin{aligned}
&zBackN = 2.0 * zBack - 1.0;\\
&zBackEyeS = 2.0 * zNear * zFar/(zFar + zNear - zBackN * (zFar - zNear));\\
&zFrontN = 2.0 * zFront - 1.0;\\
&zFrontEyeS = 2.0 * zNear * zFar/(zFar + zNear - zFrontN * (zFar - zNear));
\end{aligned} \tag{3.3}$$

As we found ourselves in a lack of precision because the clipping planes were not being properly adjusted to the bounding sphere of the scene, we set the distance between the VRP[1] and the camera to the diameter of the scene's bounding sphere.

$$dist = sceneRadio * 2; \tag{3.4}$$

Depending on the camera position, the thickness of the model has notable visual differences. The reason for such difference resides on the fact that the layers between the closest (to the observer) faces and the last back face of the model contain some other layers in between. As we can see in Fig. 3.2 the thickness calculated from the observer point of view gets the full distance traveled across the object instead the distance from the first and the second layer. This is because the back face that we selected was the last face of the model, i.e. the farthest face of the model, which generated incorrect thickness. So the method at that point only generated valid images for closed convex objects.

Therefore: we first fill a texture called *frontDepth* with the information of the depth at the model's front faces. Then we make the call to the first shader passing by parameter the texture with the depths and we compare it with the depth stored at the fragment. If the depth stored at the fragment is less or equal than the stored in the texture (plus an extra value epsilon set to 1e-6) the fragment is discarded. Afterwards, we compute the correct depth values for arbitrary cosed objects. Note that this corresponds to the first two layers of a Layered Depth Image (LDI). The full code of this first step can be found in the App. B.

The smoothing

We found many artifacts located most of the times at the borders of the model (Fig,3.3). These artifacts were caused because the edges on the second layer were colored too sharp due to the fact that the second layer is farther from the observer and it had less light contribution, thereby causing rough changes going from a smooth color to a very contrasted one. Trying to get rid of the visible artifacts on the model, in our second step we first implemented the mipmapping technique.

By using mipmapping the surface looked blurred but the images had very poor quality. This is cause because mipmapping technique gathers samples of pixels around a point and interpolates the color of the neighboring pixels. When using Mipmap, as we decrease the level of

[1]View Reference Point

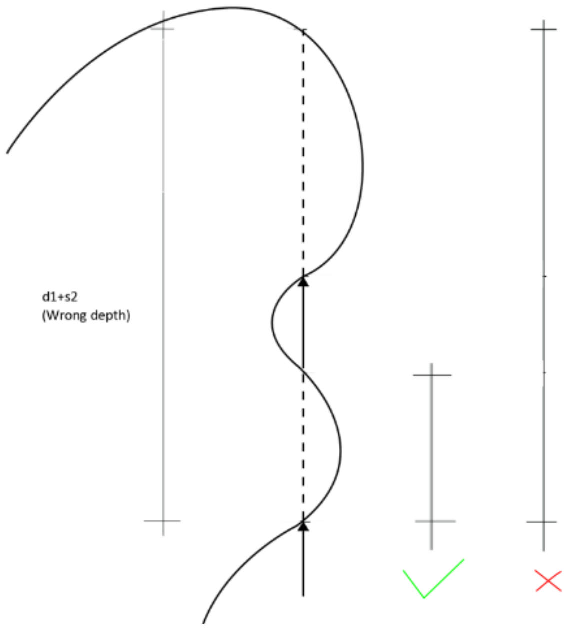


Figure 3.2: The thickness passed to the shader in this step was incorrect because both layers were being taken into account for the final thickness computation.

detail, the number of samples for the color interpolation is higher and it was producing artifacts at the model's boundaries because when performing the average, the pixels corresponding to the background were also being computed causing blank stripes artifacts. Fig. 3.4 shows an example of the poor quality image generated using mipmapping. Thus, we discarded the smoothing of the model by applying such technique, instead we used a Gaussian Kernel filter which increased the quality of the image considerably

We implemented the second pass shader smoothing method which blurred the back depth stored in the texture. We modified the Gaussian filter for applying the computations only to specific z values as follows:

When testing the Gaussian Kernel filter on the images, we noticed that nine 3x3 samples of kernel were not enough to blur the image. For that reason we finally implemented the Gaussian filter with a 5x5 Kernel. Then, the depth image looked blurred enough for keeping forward with our algorithm. A sample of the results applying our algorithm with and without gaussian Kernel can be found in detail in the Fig.3.5. And the full explanation of how we perform the Gaussian Blur and how we enhance its performance has been detailed in Appendix C.

Light Settings

Lights take a very important role in this project, specially the specular component. Many studies state [FB05, FJB04] that the more specular highlights the model has, the more translucent it looks like. Therefore, our goal is to increase the glossiness, trying to achieve the effect of, somehow, a varnish finish. Fig. 3.6 contains an example of it. For that reason we implemented

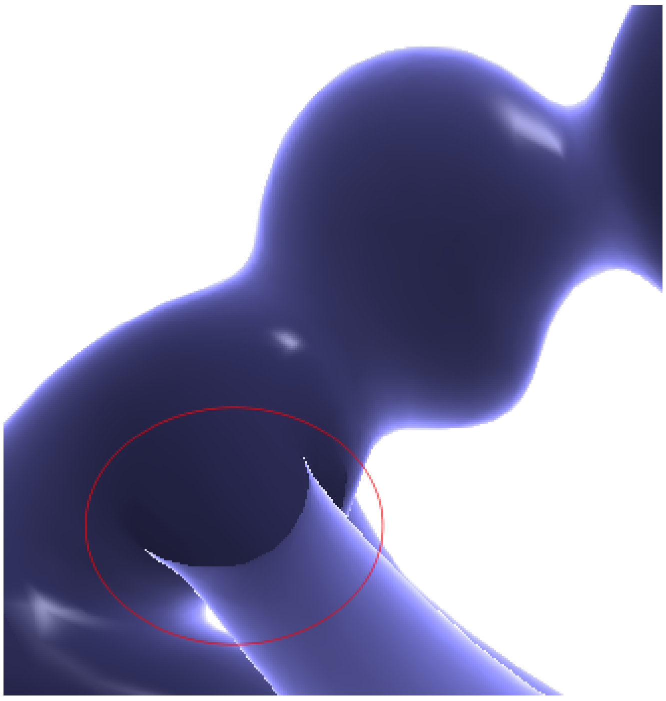


Figure 3.3: A sample of the artifacts generated by computing the exact thickness along the new direction.

the shader-based *Blinn-Phong*[2] technique which works per fragment.

The steps we reproduce are as follows: We first transform the normal into eye space and normalize the result. On the next step, we normalize the light direction. Once we have the normal in eye space and the light's direction normalized, we extract the half-vector of the light source and normalize it too. Afterwards, we compute the diffuse ambient and global ambient terms, having into consideration that there is an Ambient Occlusion parameter which has been passed to the vertex shader as an attribute, previously computed in the CPU.

We can find the full implementation of the Blinn-Phong technique in the second pass Vertex Shader found in App. C.

Ambient Occlusion Term The AO term is calculated at the beginning of the process, when we load the model. It first calculates a bounding box hierarchy in order to enhance the process for calculating the intersection of the rays for the AO process. The next step calculates the Ambient Occlusion value per vertex and store it in each of the models vertices. In order to make the process faster, we store the AO occlusion data per vertex in a file inside the project. By doing so, each time a model is loaded, if the project finds its data file with the AO values it will load it and will save some seconds (depending on the amount of vertices of the model).

[2]Blinn-Phong technique calculates a *halfway* vector between the viewer and the light-source vectors that avoids us to recalculate the angle $R \cdot V$ between a viewer and the beam from a light-source reflected on a surface.

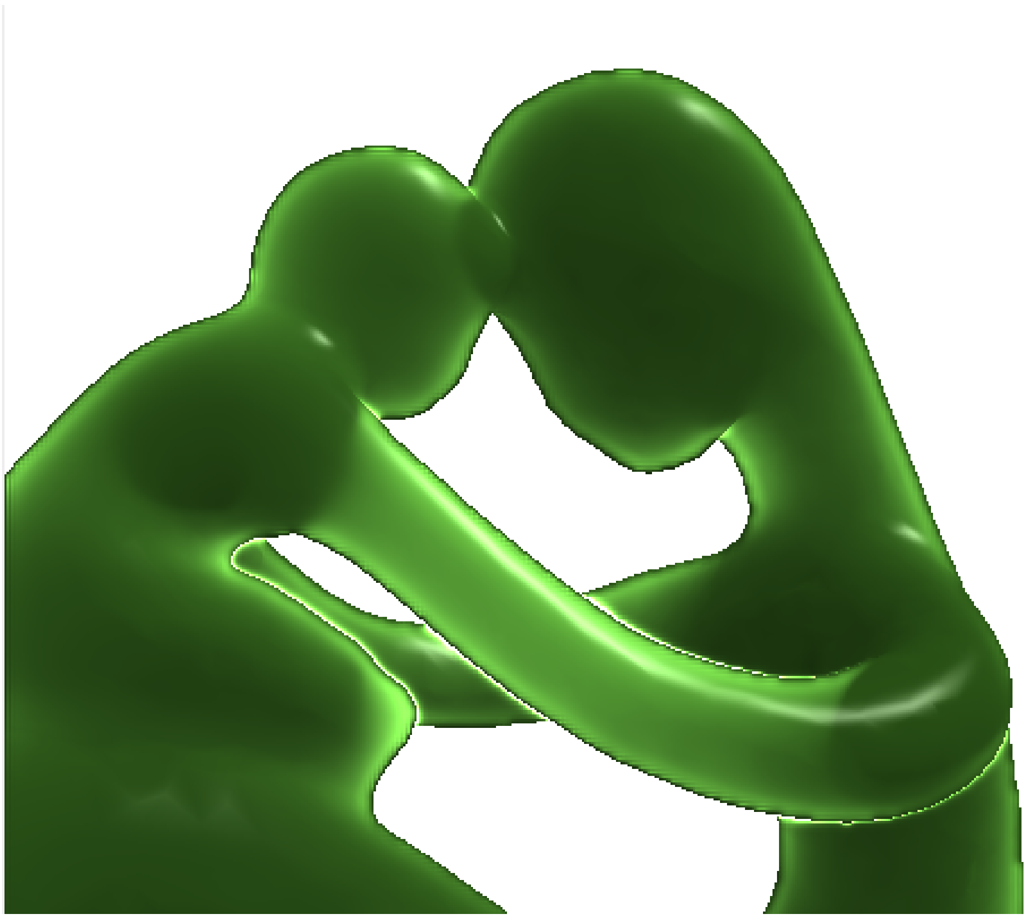


Figure 3.4: An image generated by our algorithm using Mipmap technique to filter out high frequencies. Note the blank stripes on the boundaries.

The Ambient Occlusion values have been calculated by casting rays (by default 320) in every direction of the hemisphere located at the center of the vertex which is oriented according the vertex normal. For each ray (P, w) we determine whether it intersects with an object or not, Equation 3.5 shows the equation used.

$$AO(P) = \frac{1}{nRays} \sum_{i=1}^{nRays} V(P, \omega_i) \tag{3.5}$$

The function $V(P, \omega_i)$ is 0.0 if intersects the scene, otherwise it will return 1.0. Rays reaching the background increase the brightness of the surface, whereas a ray which hits any other object contributes no illumination.

3.1.1 The Lighting Equation

The Lighting equation that we propose is as follows:

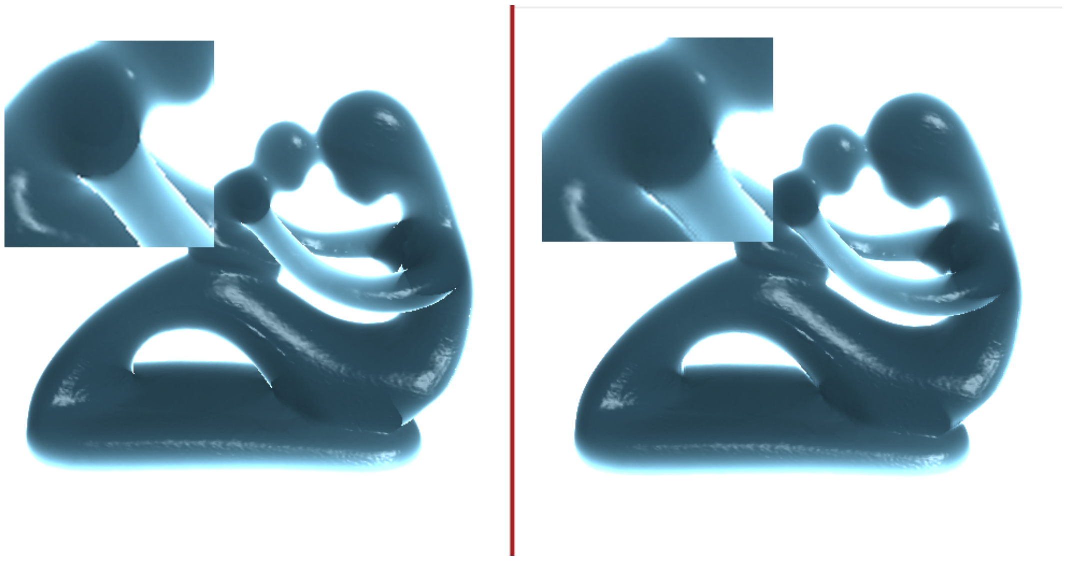


Figure 3.5: The image on the left does not have the Gaussian Kernel activated whilst the one on the right does.

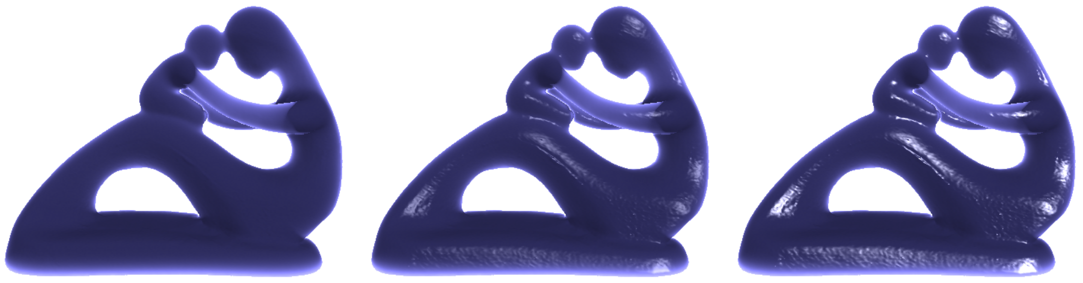


Figure 3.6: The specular highlighting of the final image will affect directly on the perception of translucency as we can see in these three images that (from left to right) have no specular component to a very high value of it.

$$\begin{aligned} glColor = & a \cdot k_a \cdot K_d \cdot I_a + \\ & w_d \cdot K_d \cdot I_d \cdot (N \cdot L) + \\ & w_s \cdot k_s \cdot I_s \cdot (N \cdot H)^n + w_b \cdot (1-t)^r \cdot (K_d + K_b) \end{aligned} \tag{3.6}$$

Blinn-Phong lighting model calculates the specular term slightly differently. Instead of calculating the angle between the view vector and the reflection vector, it calculates the half-way vector between the view and the light vector. Just to clarify the terms separately we can be subdivide it in three (3) sub-formulas. Let us start by explaining the ambient term calculation:

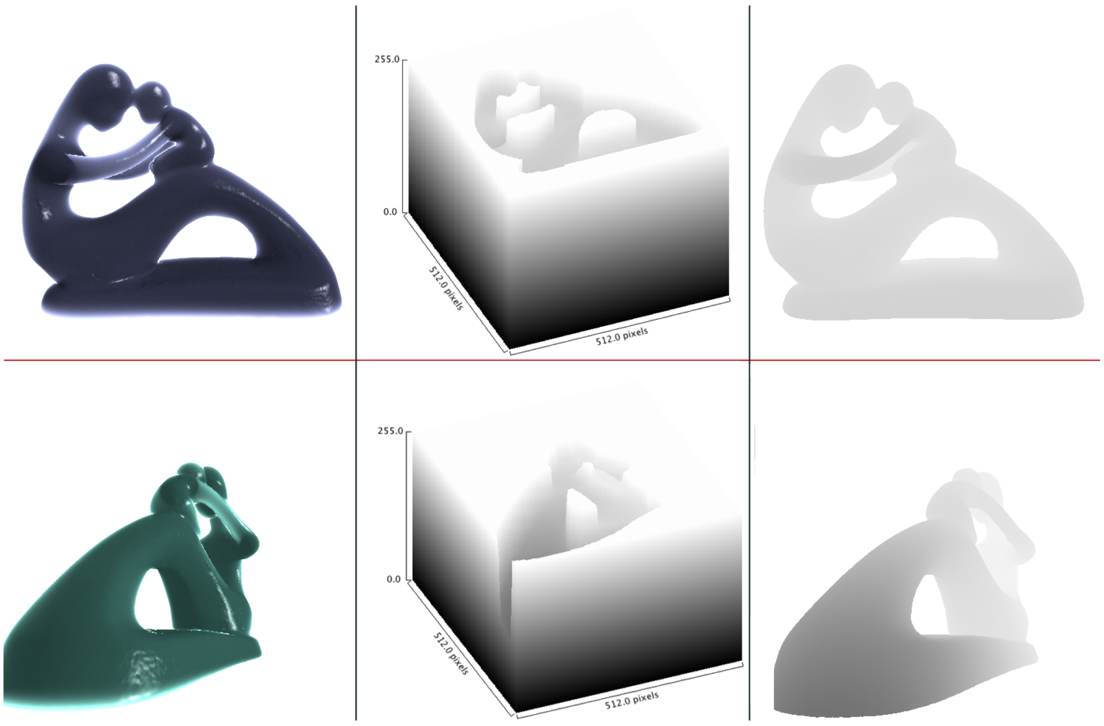


Figure 3.7: These are images extracted from the depth map of the model in different camera positions. From left to right, the images show the original image under our algorithm, the surface analized and the depth map. The surface analysis tests were performed by using the software ImageJ which can be found in the website of ImageJ.

Ambient term

$$a \cdot k_a \cdot K_d \cdot I_a \tag{3.7}$$

Where

- a is a scalar value representing the per vertex Ambient Occlusion value passed as an attribute to the vertex shader of the first step VS. This is multiplied by the ambient component of the material k_a, the ambient component of the light source I_a and the material albedo, K_d.
- k_a is a scalar value that contains the material's ambient intensity, it approximates light coming from a surface due to all the non-directional ambient light that is in the environment.
- K_d is a vector value that specifies the material's diffuse component, is the portion of the reflected intensity distribution, the *albedo* of the material.
- I_a is a vector value representing the ambient component of the scene's light source.

Diffuse term

$$w_d \cdot K_d \cdot I_d \cdot (N \cdot L) \tag{3.8}$$

Where:

- w_d Represents the diffuse contribution of the final object. It is a scalar value that can modified in execution time (as the diffuse term mostly depends on the surface normal) on the UI. By increasing this value we will see the high frequency details of the model.
- I_d is a vector value specifying the light's diffuse intensity.
- $N \cdot L$ Is a scalar value containing the maximum value of the dot product between, the normalized normal and the light's direction, and the value 0.0. Note that the advantage of the Blinn-Phong model becomes apparent if we consider both the light vector L and the view vector to be at infinity. For directional lights, the light vector L is simply the negated direction vector of the light. The Fig. 3.8 shows a global vision of the Blinn-Phong technique.

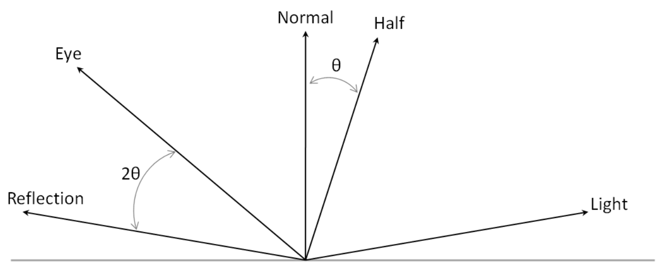


Figure 3.8: An scheme of the Blinn-Phong technique.

Specular term

$$w_s \cdot k_s \cdot I_s \cdot (N \cdot H)^n \tag{3.9}$$

Where:

- w_s Is a scalar value representing the specular reflection constant for the material. This value can also be modified in execution time from the UI.
- k_s Is a scalar value containing the material's specular intensity.
- I_s Is a vector value specifying the light source's specular intensity.
- $N \cdot H$ This is the maximum value of the dot product between the normalized normal and the half-vector parameter, and the value 0.0. The half vector is the unit vector at the half angle, i.e. angle bisector, between the eye and the light vector.
- n This is a scalar value containing the material's shininess, this value can also be modified from the UI in execution time.

Thickness term

This section includes the values to be applied to the diffuse term depending on the fragment's thickness.

$$w_b \cdot (1-t)^r \cdot (K_d + K_b) \tag{3.10}$$

Where:

- t Is a scalar value representing the fragment's thickness value. This value has been computed in real time in the fragment shader on the second step, its range is a value between 0.0 and 1.0.
- w_b Is a scalar value specifying the backlight contribution multiplied by the fragment's thickness powered to r multiplied by the diffuse component of the material K_d plus K_b which is the *diffuse weight*. The *diffuse weight* and w_b parameters can be modified in execution time from the UI.
- r Is a scalar value representing a multiplier for the thickness exponent. It will multiply the value of the thickness. We use this non-linear function because we want the backlight enhanced specially in the thinner parts of the model.
- K_d is a vector value representing the material's diffuse intensity.

CHAPTER 4

Results

4.1 Performance

The main goal for the project is to achieve interactive rates. All the tests have been performed on an Intel Core 2 Duo at 2,53GHz. The graphics card is a NVIDIA GeForce 9400M PCI 265MB. The size of the viewport in all tests was 512x512 in windowed mode.

The very first task in the project that requires a small period of time to be done is the creation of the bounding box hierarchy created for computing the rays intersection. That calculation was done in order to accelerate the ray intersection test of the Ambient Occlusion technique as it was set up with 320 rays to be intersected. This data could be saved in a file preventing us from calculating the tree every time we launch the application.
The pre-computation of this hierarchy tree is performed by CPU and takes about 3 seconds on a +240K faces model, the full information of the time took to generate the models used in our method has been detailed in Table 4.1.

Table 4.1: Bounding box creation hierarchy performance

Model	#Faces	#Vertices	Time(ms)
Fertility	+240K	480K	3486
Fertility	+120K	60K	748
Fertility	+60K	30K	353
Fertility	+30K	15K	167
Buddha	100K	50K	653

We manage to achieve interactive framerates by using only empiric ways for simulating the translucency of the objects. Table 4.2 shows statistics of the performance running our algorithm with a single light source.

The performance, as we can see in the Table 4.2, allow us to interactively render the model under different configurations in order to find the translucent look of the objects.

Table 4.2: Performance

Model	#Faces	FPS (Shaders On)	FPS (Direct light only)
Fertility	+240K	2-3	8-10
Fertility	+120K	6-7	30
Buddha	+100K	6-7	40
Fertility	+60K	8-10	53
Fertility	+30K	11-13	60
Monkey	+4K	25-27	62

4.2 Output Images

For the tests we used the Stanford's model Happy Buddha, the Blender's Monkey model and the Fertility model extracted from the Aim@SHAPE Shape Repository[1].

The variation of the color hue takes significant importance on the final perception of the translucent effect. Figures 4.1 and 4.2 show the notable difference by rendering the models with direct light only and comparing them with the renders generated by our algorithm.

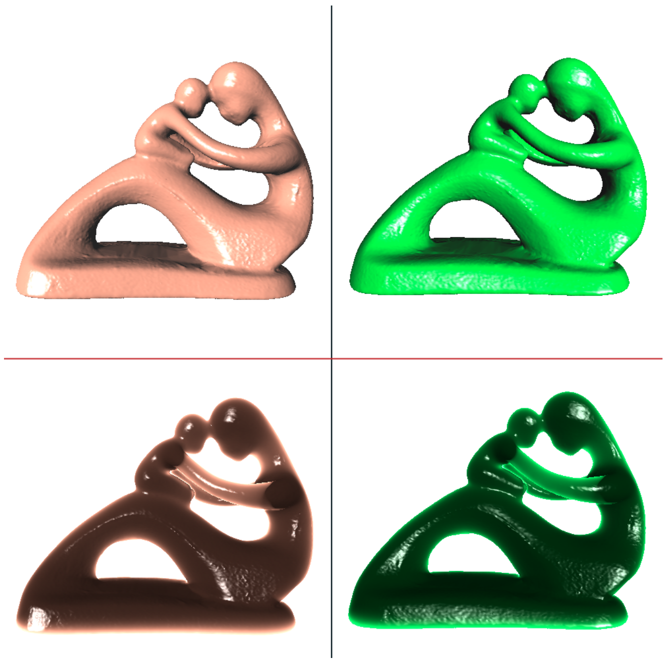


Figure 4.1: Fertility models on the upper row were rendered with direct light, the bottom row show the results by using our algorithm.

Chapter 3 contains information about the shader parameters used by our algorithm. The combination and modification of some of them generated different results affecting the visual system to perceive the model in different ways, i.e. translucent, realistic and varnished.

[1]The model was scanned with a Roland LPX-250 laser range scanner

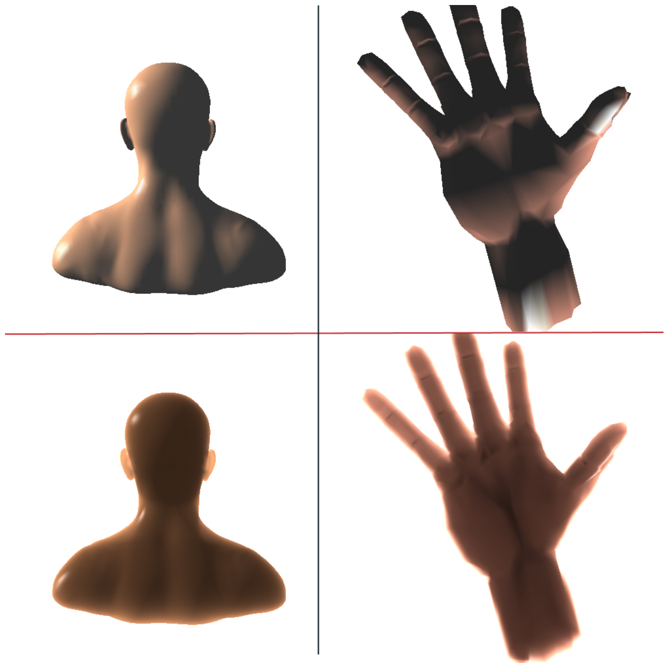


Figure 4.2: The head model and the hand on the upper row were rendered with direct light, the bottom row show the results by using our algorithm.

We performed a public survey[2] in which the participants had to answer 15 questions attached to 15 different images. Nine of these fifteen images were generated by our algorithm, the other six images, which were rendered with offline methods, were extracted from the paper published by Adolfo et al. in [MES+11]. The user study purpose is to know which parameters affect the translucent perception. The question proposed the participants to evaluate the images in a 7-levels Likert scale (from completely disagree to completely agree) on how translucent, realistic or varnished the images shown looked like.

Figure 4.3 and Figure 4.4 show the images used on the survey. The parameters modified were the *Thickness exponent*, *BackLight contribution* and *Diffuse weight*.

In the first row (purple) of our images we modified the *Thickness exponent*. The second row (green) show the images generated by modifying the *Backlight Contribution*. Finally, the third row (blue) was generated by modifying the *Diffuse weigth*. Adolfo's images in the Fig.4.4 were not modified. We only had to took away the small hint located at the left top corner of their images, where they compared their materials with real-life objects.

4.2.1 Survey statistics

This survey had a total of 44 started surveys and 38 (86,4%) completed surveys. It was done to different groups of people with different backgrounds (programmers, artists, project managers, architects, lawyers and other professionals) and the age range was 20-35.

The quality of our method can be judged from comparing the images of our algorithm in

[2]Survey performed by using Survey Monkey at SurveyMonkey.

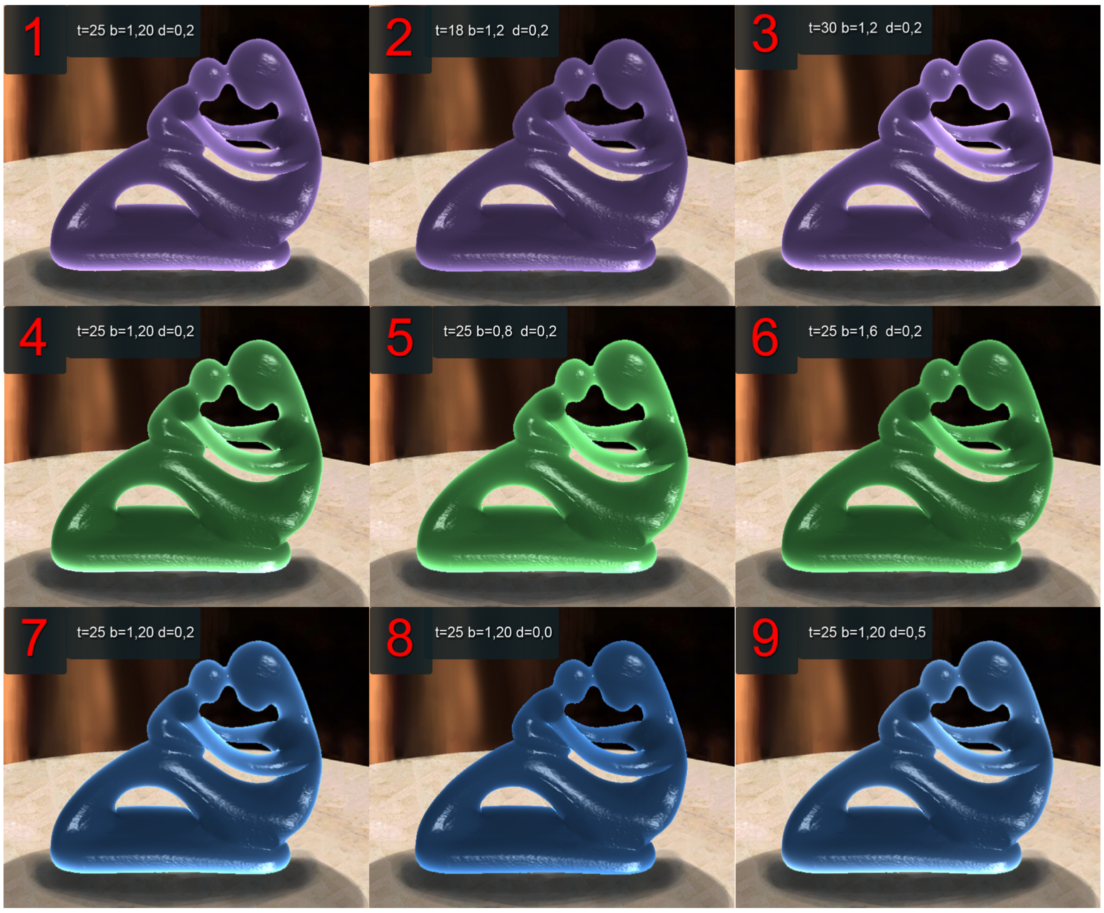


Figure 4.3: Images generated by our algorithm used in the survey, the values used for each configuration is detailed in 4.3.

Table 4.3: Parameters used in each image.

# Image	Thickness Exp. (t)	Backlight Contrib. (b)	Diffuse Weight (d)
1	25	1,20	0,20
2	18	1,20	0,20
3	30	1,20	0,20
4	25	1,20	0,20
5	25	0,80	0,20
6	25	1,60	0,20
7	25	1,20	0,20
8	25	1,20	0,0
9	25	1,20	0,50

Fig. 4.3 and the images extracted from Adolfo's study in [MES$^+$11] in Fig. 4.4, which were generated by a non-interactive technique.

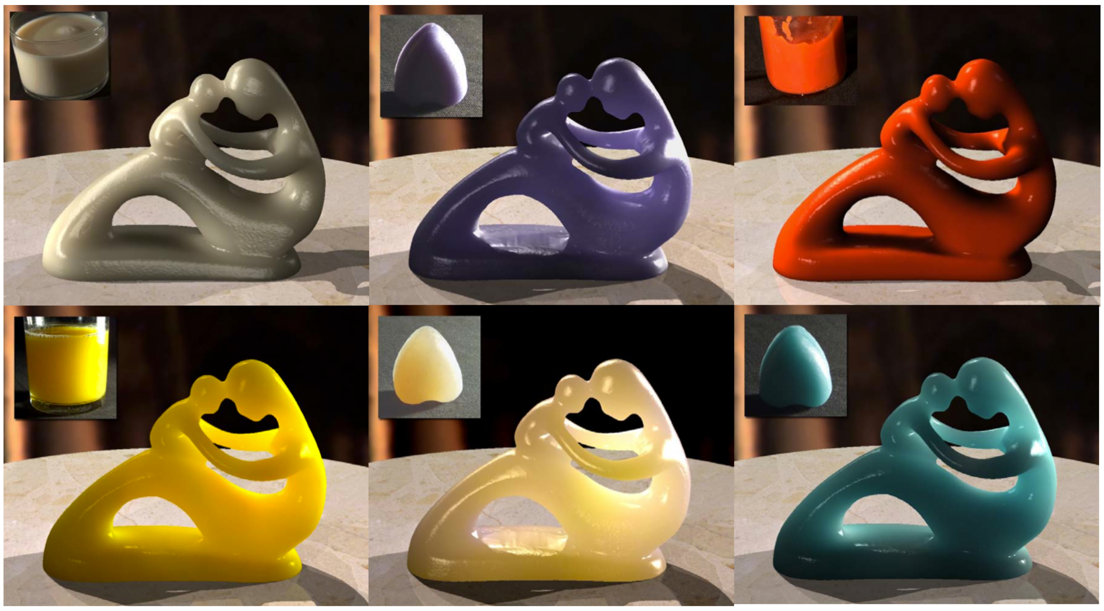


Figure 4.4: Adolfo's images used in the survey extracted from [MES+11].

Plots analysis

The Fig.4.5 includes all the plots analyzed[3] for the results gathered in the survey. The Figure has been structured as follows: The upper row contains the information of the Translucency, Realism and Varnish finish effect analysis after the modification of the *Thickness exponent* shader parameter. The second row shows the same information under the *Backlight Contribution* parameter, whilst the third row will give us a perspective of that information varying the parameter *Diffuse Weigth*.

The results of the survey have been analyzed statistically for determining any possible significance. Table 4.4 contains the analyzed statistics. The first column shows the configuration used for the render, the second row contains the *p-value* and the third column shows the F value. In addition, an extra configuration has been added to the table, such configuration was performed with an ANOVA test with 4 samples (all the other configurations were performed with 3 samples). We built the last configuration as follows: The first sample contains the data of the best image generated by our algorithm, the second column contains the less translucent image extracted from Adolfo's study. The third sample has the statistics for one *standard* Adolfo's image and the fourth sample has the information of the best Adolfo's image. All the comparison images were extracted by the publication [MES+11].

Fig. 4.6 shows the plot extracted by comparing the Adolfo's images with our results.

Let us see the Tukey HSD Test for the analysis of the last configuration (Adolfo's algorithm vs Our algorithm).

[3]All the plots were created using the portable command-line graphing utility **GnuPlot**.

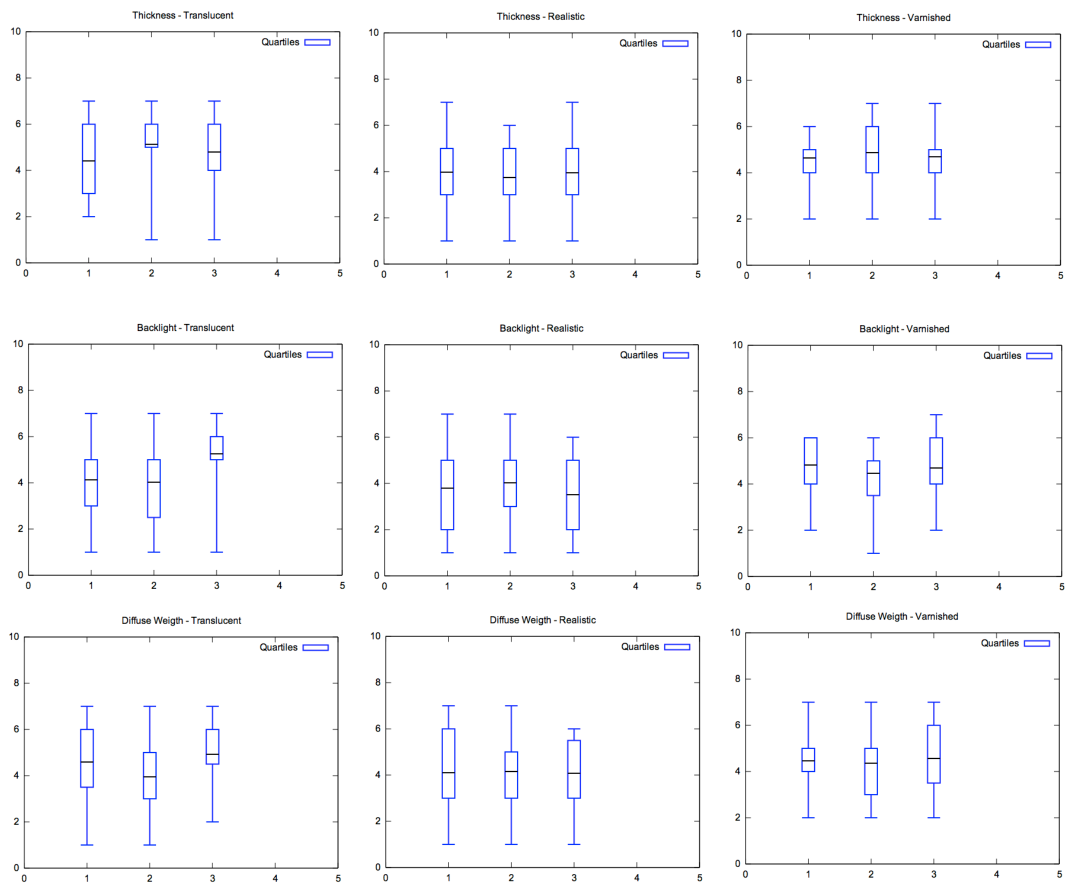


Figure 4.5: A summary of the plots that analyze each shader component and its reaction on the respondents.

Table 4.4: Results of the One-way **ANOVA**. (Significance level 10%)

Configuration	*p-value*	*F*
Realistic vs Thickness	0.087	2.52
Translucent vs Thickness	$< .0001$	22.43
Varnish finish vs Thickness	0.093	2.45
Realistic vs Backlight	0.5197	0.66
Translucent vs Backlight	0.0012	7.27
Varnish finish vs Backlight	0.398	0.93
Realistic vs Diffuse W.	0.887	0.12
Translucent vs Diffuse W.	$< .0001$	9.62
Varnish finish vs Diffuse W.	0.423	0.87
Adolfo vs Gascons	$< .0001$	59.59

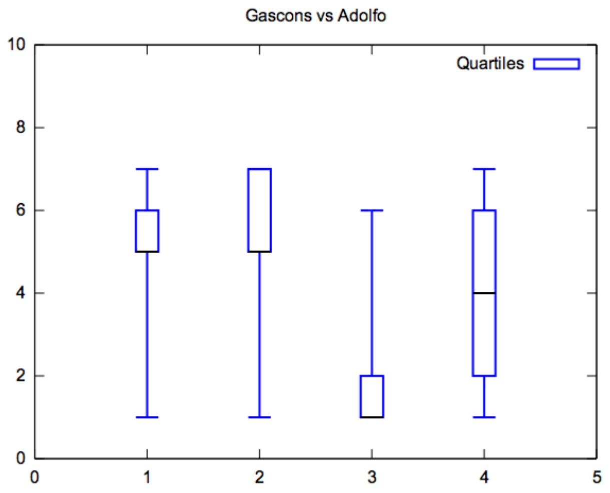


Figure 4.6: The plot comparing the results of one of the images of our algorithm and three images extracted from [MES+11].

Tukey HSD Test (Honestly Significant Difference) In this Tukey Test[4], HSD is the absolute [unsigned] difference between any two sample means required for significance at the designated level. HSD[.05] for the .05 level and HSD[.01] for the .01 level. M1 refers to the mean of Sample 1, M2: to the mean of Sample 2 and so on.
The Table 4.5 shows specifically the result of this test for the comparison of the Adolfo's images and our generated images.

Table 4.5: Tukey HSD Test (HSD[.05]=0.82; HSD[.01]=1)

Samples Configuration	P
M1 vs M2	Non significant
M1 vs M3	$P < .01$
M1 vs M4	Non significant
M2 vs M3	$P < .01$
M2 vs M4	$P < .05$
M3 vs M4	$P < .01$

From all this information we state the the *Thickness exponent* parameter was found to have a significant effect on:

(a) Translucency perception ($F = 22.43, p < 0.001$), with large values of the thickness exponent leading to images perceived as more translucent.

(b) Realistic judgement ($F = 2.52, p \simeq 0.09$), with low values of the thickness exponent leading to images perceived as slightly more realistic.

[4]Tukey HSD test is a single-step multiple comparison procedure and statistical test generally used in conjunction with an ANOVA to find which means are significantly different from one another. Named after John Tukey, it compares all possible pairs of means, and is based on a studentized range distribution q.

(c) Varnish finish perception ($F = 2.45, p \simeq 0.09$), with average values leading to images looking slightly more varnished than those generated with extreme values.

An explanation for the behavior in (a) is that the higher the *Thickness exponent*, the faster decays the contribution of the backlight with the thickness; this causes high exponent values to highlight the thinner details of the model, assuming that they will have more backlight contribution, compared with the thicker areas.

The *Backlight contribution* was found to have a significant effect on translucency perception ($F = 7.27, p \simeq 0.01$), with large values of this parameter leading to images perceived as less translucent. We found no significant effects of backlight contribution on realistic perception ($p \simeq 0.51$) and varnish finish ($p \simeq 0.39$). This drives us to conclude that too much backlight reduces the image contrast, as well as the contrast between the thinner and thicker areas of the 3D model.

Finally, the *Diffuse weight* was found to have a significant effect on translucency perception ($F = 9.62, p < 0.001$). Completely removing the diffuse term resulted in images perceived as less translucent probably because of the high-frequency detail elimination. We found no significant effects of the diffuse weight on realistic perception ($p \simeq 0.88$) and varnish finish ($p \simeq 0.42$).

The last section of the survey was done to compare the images generated with our algorithm and the ones extracted from [MES$^+$11] which where rendered in offline mode. The conclusions are satisfactory: our technique with parameters optimized for translucency lead to images which were perceived as much translucent as those generated with a high-quality, off-line subsurface scattering ray-tracer ($p < 0.01$).

CHAPTER 5

Conclusions

5.1 Conclusions

In this document we have presented a fast, GPU based empiric technique for the translucent perception of materials. We have performed a user study and the results were analyzed with One-Way **ANOVA** tests with a significance level of 10%.

Some key ingredients of our approach include the use of pre-computed per-vertex ambient occlusion, the computation of the object's thickness in screen-space through depth maps, and the use of Gaussian blur to smooth thickness before it is used in the proposed lighting equation.

Its results conclude that: with large values of the *Thickness exponent* parameter of our algorithm the translucency perception increases substantially, leading to images perceived as more translucent. The explanation for this behavior is that, the thinner parts of the model are directly affected on this parameter and the more thin they are, the more highlighted they look like. There is a clear trade-off regarding the translucency and the realistic perceptions because low values of the *Thickness exponent* lead to perceive the images as more realistic whilst high values of this parameter increases the perception of translucency. The *Diffuse Weight* also have a significant effect on the translucency perception but we found no effects of this parameter on realistic perception and varnish finish.

We have demonstrated that the translucent simulation of the objects can be merely empiric and we have generated results in a par with some of the already existing pre-processed techniques. Our results can be comparable with some of the images generated by using non-interactive rates algorithms, such as the published in [MES$^+$11].

Our main limitation is that under certain lighting and camera configurations, our method seems to generate hard shaped changes on the models materials that affect the perception of the translucency as the visual system tends to expect translucent objects to smoothly merge colors. Despite this limitation, our algorithm produces images perceived as translucent by only using empirical shading models.

APPENDIX A

User Interface

A.1 The interface

The user interface has been split into four sections: Lights, Material, Camera and Shader. Fig. A.1 shows how the User Interface looks like.

A.1.1 Lights

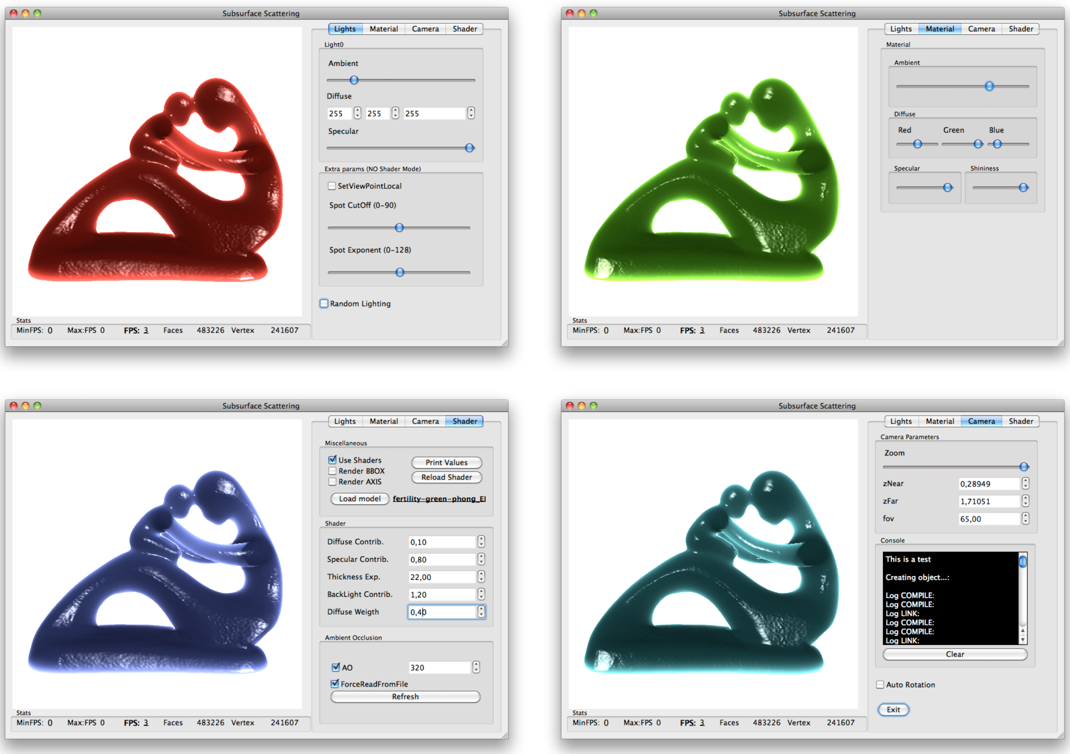
Lights section includes two sliders for the Ambient and Specular components of the light and three spin boxes only for the diffuse component (R,G,B). In addition there is a special section for setting the view point local, the light spot cut off and exponent components which can only be used when the shader is not active. Finally, a checkbox will allow us to set a *Random Lighting* mode. Such mode will modify the light position in real-time so we can see the different effects that the models will show depending on the light scenarios.

Right-clicking the scene and moving the mouse across the scene will move the light source in real-time, therefore we will be able to appreciate how to perceive the translucency depending on the light source. We also have the option of enabling a random lighting mode that makes the light source position to be moved randomly across the scene.

A.1.2 Materials

The **Materials** section has a similar appearance. It contains one slider for the Ambient, and Specular material components, and three sliders only for the Diffuse component (R,G,B). As well as the standard material components, an extra slider has been added in order to control the material shininess.

It has been proven that the best translucent effects are achieved with the highest shininess exponent but anyway has been left in the UI in order to let the user test different material conditions.



Diffuse 255 255 255

☐ SetViewPointLocal

☐ Random Lighting

MinFPS: 0 Max:FPS 0 FPS: 3 Faces 483226 Vertex 241607

☑ Use Shaders
☐ Render BBOX
☐ Render AXIS

fertility-green-phong_El

Diffuse Contrib.	0,10
Specular Contrib.	0,80
Thickness Exp.	22,00
BackLight Contrib.	1,20
Diffuse Weigth	0,40

☑ AO 320

☑ ForceReadFromFile

zNear	0,28949
zFar	1,71051
fov	65,00

```
This is a test

Creating object...:

Log COMPILE:
Log COMPILE:
Log LINK:
Log COMPILE:
Log COMPILE:
Log LINK:
```

☐ Auto Rotation

Figure A.1: A screenshot of all the UI tabs.

A.1.3 Camera

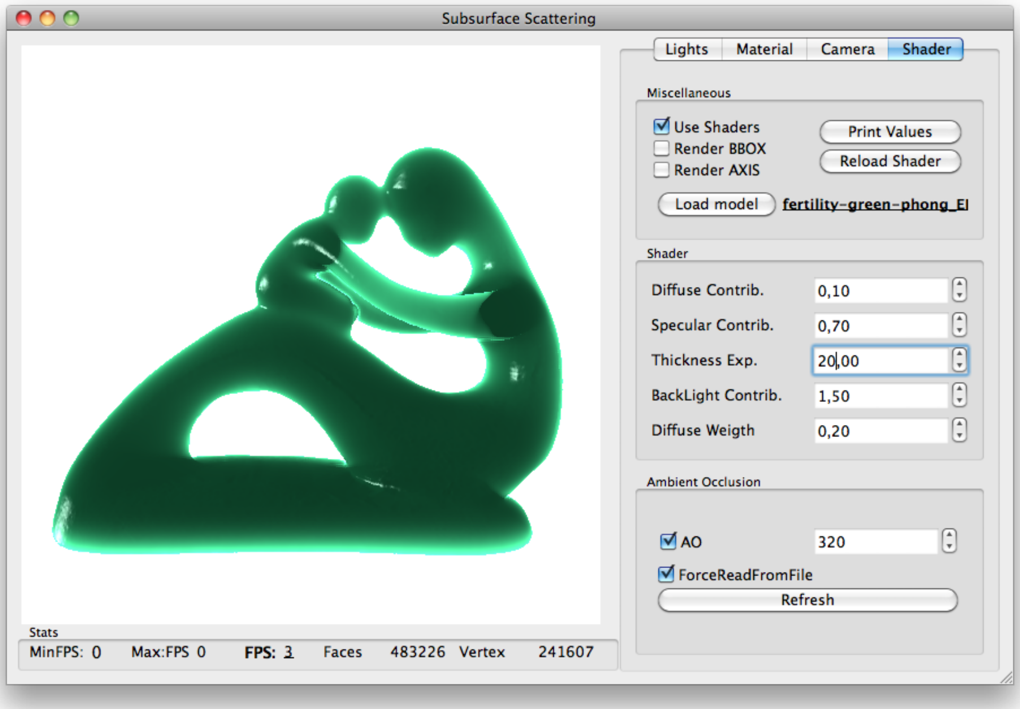
This section is in charge of the camera parameters setting. It contains a slider for the Zoom, two spin boxes for the clipping planes and one more for the field of view as well as a console for checking any parameter or message error. It has also a checkbox called *Auto Rotation* which will enable a rotation fixed on the Y axis.

A.1.4 Shader

The last and more important section of the User Interface is the **Shader** section. It has been split into three subsections: Miscellaneous, Shader and Ambient Occlusion.

In the Miscellaneous section we control the activation or deactivation the shaders, we can find too checkBoxes for activating the rendering of the bounding box and the Coordinates Axis. It also includes a button which has the functionality of printing all the current data, a button for reloading the shader (in case we do not want to re-launch again the application after modifying the shader code). This section will give us too the opportunity of loading a different 3D model by clicking on the Load Model button.

The Shader section is the core of the application, the shader parameters are controlled from this section of the User Interface. These parameters are explained in detail in the Appendix C. A screenshot of the UI with the Shader tab selected is shown in Fig. A.2.



[✓ Use Shaders]
Render BBOX
Render AXIS
fertility-green-phong_El

Diffuse Contrib.	0,10
Specular Contrib.	0,70
Thickness Exp.	20,00
BackLight Contrib.	1,50
Diffuse Weigth	0,20

[✓ AO] 320
[✓ ForceReadFromFile]

MinFPS: 0 Max:FPS 0 FPS: 3 Faces 483226 Vertex 241607

Figure A.2: A screenshot of the UI with the *Shader* tab selected.

At the bottom of the screen (underneath the viewport) we can find a status bar, which will be updating the information regarding: the minimum, maximum and average frame rate tracked, as well as the number of faces and vertex of the 3D model loaded.
We conclude this Appendix with the AO parameters section. We will be able to activate or deactivate this effect in execution time, setting the number of rays to trace per vertex.

APPENDIX B

First Pass Shader

B.1 Fragment Shader

The main functionality of the first pass shader is to get the depth values of the second layer. We perform such operation by discarding those fragments whose depth is greater than the front faces depth stored in the texture filled before calling the shader.

The interesting code can be found in the fragment shader, which has been divided in three simple steps. These are:

1. Get the depth stored on the texture passed to the vertex shader called *frontDepth* which will contain the depth values of the model's first layer (this texture is filled before calling to this shader).

2. Get the fragment's depth and store it in the variable *zFront*.

3. Compare the depths. If the *zFront* value is less or equal of the *frontDepth* value (with the addition of a very small *epsilon* component) the fragment will be discarded.

The Fig. B.1 will help us to understand this shader functionality by showing an image where, from the observer point of view, we select the proper layers to be used on computing the model's thickness.

```
uniform sampler2D frontDepth;
const float epsilon = 1e-6;

void main()
{
    float zFrontTexture = texture2D(frontDepth,gl_FragCoord.xy/512.0).x;
    float zFront = gl_FragCoord.z;
    if(zFront <= zFrontTexture+epsilon)
      discard;
}
```

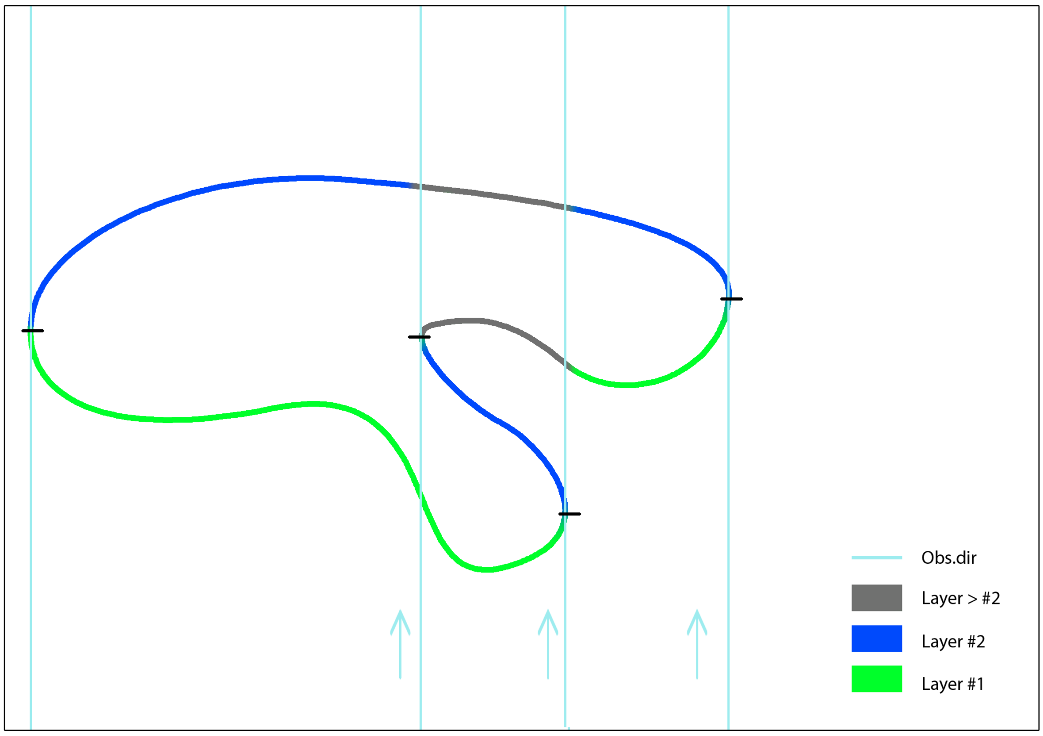


Figure B.1: This figure shows the layers that our algorithm will take for the computation of the final thickness.

B.1.1 Variables explanation

- **zFront**: The depth value in the current fragment.
- **zFrontTexture**: Depth value of the *frontDepth* texture for that fragment.
- **frontDepth**: The texture that contains the depth values of the first layer.

APPENDIX C

Second Pass Shader

Let us start with the Vertex Shader. The purpose of it is to calculate the half vector for implementing the Blinn-Phong technique, compute the diffuse and ambient terms, gather the AO scalar value passed by parameters and finally send the information to the FS.

C.1 Vertex shader

```
attribute float ambientOcclusionPerVertex;
varying float ambientOcclusion;

varying vec4 diffuse,ambient;
varying vec3 normal,lightDir,halfVector;

void main()
{
        /* Transform the normal into eye space and normalize it */
        normal = normalize(gl_NormalMatrix * gl_Normal);

        /* Normalize the light's direction (the light
        is stored in eye space). The position field is actually
            direction */
        lightDir = normalize(vec3(gl_LightSource[0].position));

        /* Normalize the halfVector to pass it to the fragment shader */
        halfVector = normalize(gl_LightSource[0].halfVector.xyz);

        /* Compute the diffuse , ambient and globalAmbient terms */
        diffuse = gl_FrontMaterial.diffuse * gl_LightSource[0].diffuse;
        ambient = gl_FrontMaterial.ambient * (gl_LightSource[0].ambient
            + gl_LightModel.ambient);

        ambientOcclusion = ambientOcclusionPerVertex;

        gl_Position = ftransform();
        gl_FrontColor = gl_Color;
}
```

C.1.1 Variables explanation

The variables used in this vertex shader are the following:

- **ambientOcclusionPerVertex**: A scalar value representing the value of the per vertex ambient occlusion.
- **ambientOcclusion**: The varying parameter to be passed to the Fragment Shader with the information of the Ambient Occlusion.
- **diffuse**: It is a 4D varying vector representing the diffuse intensity.
- **ambient**: It is a 4D varying vector representing the ambient intensity.
- **normal**: It is a 3D varying vector representing the normal.
- **lightDir**: It is a 3D varying vector representing the light direction.
- **halfVector**: It is a 3D varying vector representing the half vector normalized..

Except the **ambientOcclusionPerVertex** value, all the variables explained here will be sent to the Fragment Shader.

C.2 Fragment Shader

Due to the fact that the following fragment shader contains the whole engine of the project, it will be split into four parts. The first part contains the variables, uniforms, attributes, etc. declarations.

C.2.1 Variables declarations

In this section of the fragment shader we declare all the necessary variables for performing the second pass.

```
uniform sampler2D backDepth;
uniform sampler2D frontDepth;

uniform float dist;
uniform float lod;
uniform float diffuseContribution;
uniform float specularContribution;
uniform float thicknessExp;
uniform float diffuseWeigth;
uniform float backLightContribution;

varying float ambientOcclusion;
varying vec4 diffuse,ambient;
varying vec3 normal,lightDir,halfVector;

#define KERNEL_SIZE_XL 25

float step = 1.0/256.0;
float kernel_XL[KERNEL_SIZE_XL];
vec2 offset_XL[KERNEL_SIZE_XL];
```

C.2.2 Smoothing technique

We have implemented a technique which samples the second depth layer using a Gaussian Filter in order to blur the back side of the texture. It receives as a parameter a 2D sampler called *sampler*. In order to make this technique efficient, we have performed some modifications to the standard Gaussian Filter algorithm, these are:

- We first calculate the offsets for the 5x5 kernel samples.
- Then we create a float variable called *depth* which will contain the depth of the *sampler* texture.
- We set to a temporary variable (*tmp*) the depth at the samples and we store also to the variable *desv* the absolute value of the subtraction between *tmp* and the depth stored in the texture.
- In order to accelerate the process of the Gaussian Blur we first discard the fragments corresponding to the background by comparing *tmp* with 0.99 (note that the depth value will be a scalar value between 0.0 and 1.0).
- Finally, as Eq. C.1 states, we assign a weight to a variable w. By clamping the deviation previously computed (*desv*) with the factors 0.0 and 1.0 we make sure that the high changes in the thickness will not be taken into account.

  $$w = kernel\_XL[i] * (1.0 - clamp(desv/0.10, 0.0, 1.0)); \tag{C.1}$$

  We invert the behavior of the clamp function by computing the subtraction between 1.0 and the result of the clamp, Fig. C.1 shows an example of it.

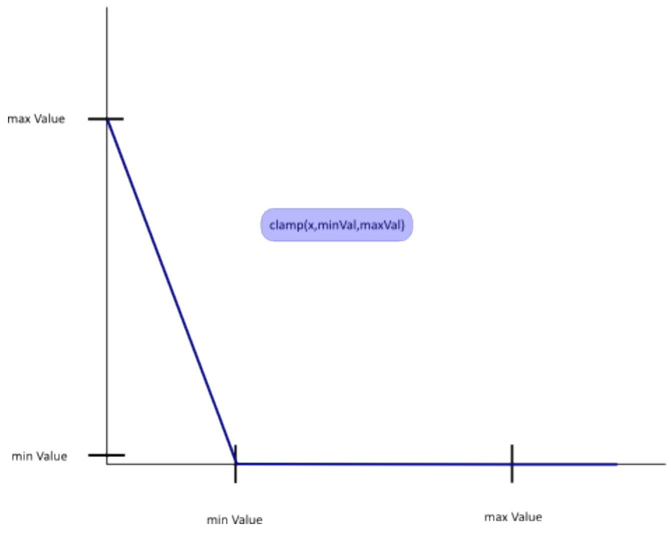


Figure C.1: This how Eq. C.1 looks like.

- Once we have the depth we will compute along all the samples the values for applying the smoothing to each fragment.

```
float smoothDepthXL(sampler2D sampler)
{

   int i = 0;
   float sum = 0.0;
   float W=0.0;

    float k[KERNEL_SIZE_XL + 1];

    k[0] = 1.0/26.;
    k[1] = 4.0/26.;
    k[2] = 7.0/26.;
    k[3] = 16.0/26.;
    k[4] = 26.0/26.;
    k[5] = 41.0/26.;

    offset_XL[0] = vec2(-2.0*step, -2.0*step);
    offset_XL[1] = vec2(-step,  -2.0*step);
    offset_XL[2] = vec2(0.0,  -2.0*step);
    offset_XL[3] = vec2(step,  -2.0*step);
    offset_XL[4] = vec2(2.0*step, -2.0*step);

    offset_XL[5] = vec2(-2.0*step,-step);
    offset_XL[6] = vec2(-step,  -step);
    offset_XL[7] = vec2(0.0, -step);
    offset_XL[8] = vec2(step, -step);
    offset_XL[9] = vec2(2.0*step, -step);

    offset_XL[10] = vec2(-2.0*step, 0.0);
    offset_XL[11] = vec2(-step, 0.0);
    offset_XL[12] = vec2(0.0, 0.0);
    offset_XL[13] = vec2(step, 0.0);
    offset_XL[14] = vec2(2.0*step, 0.0);

    offset_XL[15] = vec2(-2.0*step, step);
    offset_XL[16] = vec2(-step, step);
    offset_XL[17] = vec2(0.0, step);
    offset_XL[18] = vec2(step, step);
    offset_XL[19] = vec2(2.0*step, step);

    offset_XL[20] = vec2(-2.0*step, 2.0*step);
    offset_XL[21] = vec2(-step, 2.0*step);
    offset_XL[22] = vec2(0.0, 2.0*step);
    offset_XL[23] = vec2(step, 2.0*step);
    offset_XL[24] = vec2(2.0*step, 2.0*step);

    kernel_XL[0] = k[0]; kernel_XL[1] = k[1]; kernel_XL[2] = k[2];
        kernel_XL[3] = k[1]; kernel_XL[4] = k[0];
    kernel_XL[5] = k[1]; kernel_XL[6] = k[3]; kernel_XL[7] = k[4];
        kernel_XL[8] = k[3]; kernel_XL[9] = k[1];
    kernel_XL[10] = k[2]; kernel_XL[11] = k[4]; kernel_XL[12] = k[5];
        kernel_XL[13] = k[4]; kernel_XL[14] = k[2];
    kernel_XL[15] = k[1]; kernel_XL[16] = k[3]; kernel_XL[17] = k[4];
        kernel_XL[18] = k[3]; kernel_XL[19] = k[1];
    kernel_XL[20] = k[0]; kernel_XL[21] = k[1]; kernel_XL[22] = k[2];
        kernel_XL[23] = k[1]; kernel_XL[24] = k[0];

    float depth = texture2DLod(sampler, gl_FragCoord.xy/512.0, lod).x;
   for( i=0; i<KERNEL_SIZE_XL; i++ )
   {
       float tmp = texture2DLod(sampler, (gl_FragCoord.xy/512.0 +
           offset_XL[i]), lod).x;
        float desv = abs(tmp-depth);
        if (tmp < 0.99)
        {
            float w = kernel_XL[i] * (1.0 - clamp(desv/0.1, 0.0, 1.0));
            sum += tmp * w;
            W += w;
        }
    }

    return sum/W;

}
```

C.2.3 Lighting computations

This section is in charge of computing the per fragment *Blinn-Phong* calculations. The steps followed are:

- First, we start setting to the variable *color* the value of the AO, ambient and the diffuse components for that fragment.
- Next, we create a variable n for storing the normalized interpolated normal.
- The next step to follow is the computation of the dot product between the normal n and the light direction *lightDir*.
- If the result of the dot product is higher than 0.0 the *color* variable will be modified. The modifications include the following changes:
  - The addition to the *color* of the diffuse component with the proper direction (multiplying it for the *NdotL* variable and we increase or decrease its intensity by multiplying it with the *diffuseContribution* parameter.
  - Add to the *color* variable the specular component of the material and the light source multiplied by the *specularContribution*. The result of such combination is multiplied by the dot product between the halfVector normalized (*halfV*) and a null value (0.0) powered to the material shininess.

```
vec4 computeLight()
{
    vec3 n,halfV;
    float NdotL,NdotHV;

    vec4 color = ambientOcclusion*ambient*diffuse;
    n = normalize(normal);
    NdotL = max(dot(n,lightDir),0.0);

    if (NdotL > 0.0) {
            color += diffuseContribution*diffuse * NdotL;
            halfV = normalize(halfVector);
            NdotHV = max(dot(n,halfV),0.0);
            color += specularContribution*gl_FrontMaterial.specular *
                            gl_LightSource[0].specular *
                            pow(NdotHV, gl_FrontMaterial.shininess);
    }
    return color;
}
```

C.2.4 Main function

The last steps to conclude our Lighting Equation are:

- Get the depth values of the second layer in *zBack* and the depth of the fragment (which will be the first layer as it has been computed in the first pass of the shader explained in Appendix B).
- The second step in the *main* function is to transform the depth values into eye-space.
- *thickness* will contain the value of the distance between the first and the second layer divided by the diameter of the bounding sphere of the scene (stored in the variable *dist*).
- *backLighting* will contain the *thickness* powered to the *thicknessExp*.
- The fragment color at the end of the computations will be a mix of all the values gathered in these last steps performing the **Lighting equation** of our algorithm which was explained in detail in Chapter 3.1.1.

```
void main()
{

    float zBack = smoothDepthXL(backDepth);
    float zFront = texture2DLod(frontDepth, gl_FragCoord.xy/512.0, 0.0).
        x;
    zFront = gl_FragCoord.z;

    float A = gl_ProjectionMatrix[2].z;
    float B = gl_ProjectionMatrix[3].z;

    float zNear = - B / (1.0 - A);
    float zFar  =   B / (1.0 + A);

    float zBackNormalized = 2.0 * zBack - 1.0;
    float zBackEyeSpace = 2.0 * zNear * zFar / (zFar + zNear -
        zBackNormalized * (zFar - zNear));

    float zFrontNormalized = 2.0 * zFront - 1.0;
    float zFrontEyeSpace = 2.0 * zNear * zFar / (zFar + zNear -
        zFrontNormalized * (zFar - zNear));

    float thickness = (zBackEyeSpace - zFrontEyeSpace)/(dist);
    float backLighting = pow(1.0 - thickness, thicknessExp);

    gl_FragColor =  computeLight() + backLightContribution*backLighting
        *(diffuse + vec4(diffuseWeigth));
}
```

Bibliography

[BBNM07] Antoine Bouthors, Eric Bruneton, Fabrice Neyret, and Nelson Max, *Real-time subsurface scattering on the gpu*, 2007.

[DL06] William Donnelly and Andrew Lauritzen, *Variance shadow maps*, Proceedings of the 2006 symposium on Interactive 3D graphics and games (New York, NY, USA), I3D '06, ACM, 2006, pp. 161–165.

[FB05] Roland W. Fleming and Heinrich H. Bülthoff, *Low-level image cues in the perception of translucent materials*, ACM Trans. Appl. Percept. **2** (2005), 346–382.

[FJB04] Roland W. Fleming, Henrik Wann Jensen, and Heinrich H Bülthoff, *Perceiving translucent materials*, Proceedings of the 1st Symposium on Applied perception in graphics and visualization (New York, NY, USA), APGV '04, ACM, 2004, pp. 127–134.

[HBV03] Xuejun Hao, Thomas Baby, and Amitabh Varshney, *Interactive subsurface scattering for translucent meshes*, Proceedings of the 2003 symposium on Interactive 3D graphics (New York, NY, USA), I3D '03, ACM, 2003, pp. 75–82.

[HK93] Pat Hanrahan and Wolfgang Krueger, *Reflection from layered surfaces due to subsurface scattering*, 1993, pp. 165–174.

[JB05] Henrik Wann Jensen and Juan Buhler, *A rapid hierarchical rendering technique for translucent materials*, ACM SIGGRAPH 2005 Courses (New York, NY, USA), SIGGRAPH '05, ACM, 2005.

[JG10] Jorge Jimenez and Diego Gutierrez, *Gpu pro: Advanced rendering techniques*, ch. Screen-Space Subsurface Scattering, pp. 335–351, AK Peters Ltd., 2010.

[JSM$^+$01] Henrik Wann Jensen, Jensen Stephen, Stephen R. Marschner, Marc Levoy, and Pat Hanrahan, *A practical model for subsurface light transport*, 2001.

[MES$^+$11] Adolfo Munoz, Jose I. Echevarria, Francisco Seron, Jorge Lopez-Moreno, Mashhuda Glencross, and Diego Gutierrez, *BSSRDF estimation from single images*, Computer Graphics Forum **30** (2011), 455–464.

[MKB$^+$03] Tom Mertens, Jan Kautz, Philippe Bekaert, Frank Van Reeth, Hans-Peter Seidel, Frank Van, Reeth Hans peter Seidel, and Limburgs Universitair Centrum, *Efficient rendering of local subsurface scattering*, In Proceedings of the 11th Pacific Conference on Computer Graphics and Applications, 2003, p. 51.

[NRH$^+$92] F. E. Nicodemus, J. C. Richmond, J. J. Hsia, I. W. Ginsberg, and T. Limperis, *Radiometry*, Jones and Bartlett Publishers, Inc., , USA, 1992, pp. 94–145.

[Sta95] Jos Stam, *Multiple scattering as a diffusion process*, In Eurographics Rendering Workshop, 1995, pp. 41–50.