Human-Skin Rendering

Juan Pacheco
Digipen Institute of Technology Europe-Bilbao

Abstract

This paper contains the implementation of a Realistic Real-Time Skin rendering technique that computes how light interacts, refracts, and gets absorbed by human skin. This implementation is based on the paper explained in chapter 14 of the book GPU GEMS 3 written by Nvidia. This book adapts the original paper [d'Eon et al. 2007] that explains an efficient method to simulate realistic human skin that could be used in video games, as its performance is fast and realistic enough. This implementation contains a pipeline that divides the human skin into two main parts, surface and subsurface reflectance, and finally mixes them in a composition pass. The technique is implemented in a framework using Vulkan, and it contains an editor with customizable parameters. Also, the original 3D head asset has been taken from the 3DScanStore webpage.

1 Skin Model

As human skin is widely known to be computationally expensive to render because of the high-frequency details, cavities, freckles, etc, it contains, this approach is helped by the use of 3D scans to make it easier. But this is not enough; a physically realistic lighting model needs to be implemented to make the skin look real. Techniques such as subsurface scattering and multilayer materials need to be explored to be able to mimic its realism while maintaining the computations cheap. For that, this technique [d'Eon et al. 2007] uses a three-layer skin model compared to other ones that use five layers [Krishnaswamy and Baranoski 2004], being able to keep it suitable for Real-Time.

1.1 Skin Surface Reflectance

When light reaches the skin, usually 6% of it gets reflected completely. This happens when the light hits the first oily layer of the skin, not entering the Epidermis and the Dermis. As the skin is very rough, the reflected light will go in many directions, needing an accurate PBR BRDF model to represent this accurately, alongside a highly detailed normal map with distinguishable normals.

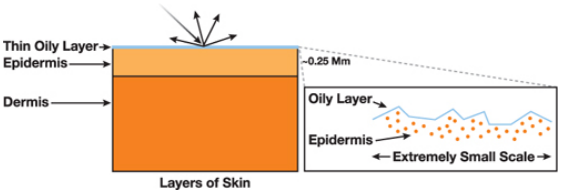

Figure 1: Skin Reflectance [d'Eon et al. 2007]

1.2 Skin Subsurface Reflectance

The phenomenon of subsurface reflectance happens when the incoming light gets absorbed, scatters, and exits the surface, or passes through the surface directly. These events give the skin its soft and colored appearance, but to make the properties work, some physical properties need to be modeled too.

One of the problems here is that each layer of the skin has different properties, so managing to compute the properties and corresponding scatter for every layer could be expensive if we overscope the amount of these. However, it has been researched

that with two layers the result is accurate enough [Donner and Jensen 2006], but as previously said, in this implementation three layers are used.

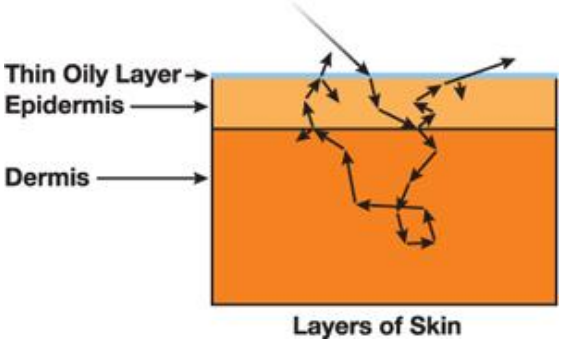

Figure 2: Skin Subsurface Reflectance [d'Eon et al. 2007]

2 Specular Reflectance

Although the Phong model could give a result that may be acceptable, it fails to create a reliable reflection at some specific angles, which is why in the implementation the specific PBR model that is used for the specular reflection is the one explained by Kelemen /Szirmay-Kalos [Kelemen et al. 2001].

3 Fresnel Reflectance

This term is something presented in all PBR models, and this one is not an exception. As the Fresnel term defines the reflection and transmission of light when it reaches a point where a media change happens, we must assume an index of refraction for the skin, in this case, 1.4 is assumed, which gives us a reflectance at a normal incidence of 0.028.
As with many other models, instead of computing the entire Fresnel Equations, the Fresnel-Schlick approximation is used because the results are almost the same and it is much cheaper computationally speaking. Also, as the skin surface tends to be very rough, the half-vector is used instead of the normal one for the computation, as it generates better results for grazing angles.

4 Beckmann distribution function

Through the years, one of the most standardized models in PBR rendering has been the Cook-Torrance model [Cook and Torrance 1982], skin needs more specific light interaction, which is why the Beckmann distribution function is used.

$$D(m) = \frac{1}{\alpha_b^2 \cdot cos(\theta_m)^4} \cdot e^{\frac{-tan(\theta_m)^2}{\alpha_b^2}}$$

However, computing this function in every frame could be significantly expensive, so the original writers decided to store the results of these values in a precomputed square texture. This texture is computed by varying the dot product between the light and half vector, and the roughness over the UV coordinates. For more optimization, these values could be packed into 8 bits and unpacked when needed.


Figure 3: Beckmann texture

5 Specular BRDF

After all the necessary components have been calculated, it is time to assemble the final BRDF equation.

$$BRDF = max(\frac{B \cdot F}{dot(h,h)}, 0)$$

Something worth mentioning is that the skin will not be adding to the color of a specular's light color, which means the reflected color will be the same as the color of the light.

6 Diffusion Profiles

Diffusion profiles are one of the most important cores in subsurface scattering to analyze how light reacts with translucent materials. In physics, it is the outcome of an experiment that consists of a white beam hitting a flat surface perpendicularly. The light that gets reflected, ergo not absorbed, in the function of the angle and distance is the result of this experiment. Every color has its own profile, and although originally some implementations compute their own profiles [Donner and Jensen 2006], this paper keeps it simpler.

Once we have the diffusion profiles, we need to collect the incoming light at every surface point and then scatter it around according to the shape of the profiles. We could also ignore the direction of the light as it diffuses very fast, however, we will need to consider the angle between the normal and the light vector. After that, the light gets scattered and exits the surface in every direction.

As I said before, to keep it simple this paper gets the computed diffusion profiles for a 3-layer skin model developed by a previous paper [Donner and Jensen 2005] and approximates the rendering of curved skin by only taking into account the distance between two points of the surface, not needing to care about the geometry that is in the middle. This result is not entirely accurate, but it is a great approximation for fast skin rendering.

Another approximation to compute the resulting diffusion profiles is to use Gaussian functions. The properties of this equation indicate that we could add two Gaussians, and the resulting function will still be a Gaussian. Because of that, the Gaussians used in this paper to approximate the 3-layer model [d'Eon et al. 2007] diffusion profiles are the following.

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

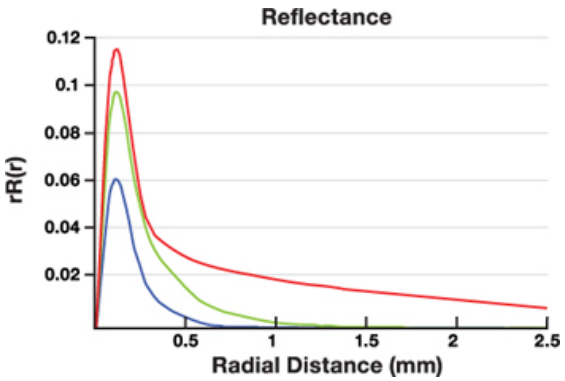

Figure 4: Diffusion Profiles [d'Eon et al. 2007]

Also, the weights for each channel add up to 1, as we want the average outgoing light to be white too, and then decide the color based on a texture.

7 Texture-Space Diffusion

This technique, applied in The Matrix sequels [Borshukov and Lewis 2003], consists of unwrapping a 3D model onto a 2D quad and performing light computations for using them later as a light map.

Figure 5: Irradiance map

After unwrapping the mesh and computing the light interaction, the result will be convoluted to simulate the scattering of the light. The shape used for the blurs is the shape of the diffusion profile, and the idea here is to have one texture per Gaussian used to get the profile, the first one being the texture of the irradiance map without blurring, so in this specific case, we will have 6 textures, hence 5 blurs. After performing all the blurs, we will combine them with the same linear combination used before to approach the Gaussians to the diffusion profile.

Figure 6: Irradiance map last blur

It is important to mention that the blurring happens in two different passes, one in the direction of U and the second one in the direction of V, and that each blur contains 7 taps, where each weight (0.006, 0.061, 0.242, 0.383, 0.061, 0.006) belongs to a Gaussian of deviation 1. Also, we will multiply this blur by the current scale of the Stretch Map described in the next section. All these blur passes are separable and possible because one of the properties of the Gaussians is that if you convolve two Gaussians the result will be a Gaussian too.

8 Stretch Map

This step would not be needed if the surfaces we are rendering did not have curvature, however, as we are rendering a head that has different curves, the distance of the surfaces in world space is not the same as the distance in texture space because of UV distortion, so a Stretch Map needs to be computed.

This Stretch Map will fix the UV distortion and will be applied when blurring the irradiance maps, and the computation of it is fairly easy. We will unwrap the 3D model as we did before, and we will only need to store the inverse of the partial derivatives of the world space coordinates to get the results in U and V. Also, a scale factor may need to be applied to fit these values into a range of [0, 1].

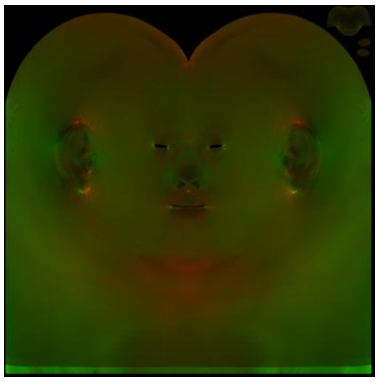

Figure 7: Stretch map

One of the things that may need to be considered when computing the Stretch Map is that the values will be constant through the triangles of the mesh, so we may need to blur these textures to avoid creating artifacts. This means that we will need to duplicate the blur passes to blur also the Stretch Maps, and then we will use the blurred Stretch Map in the following blurring pass.

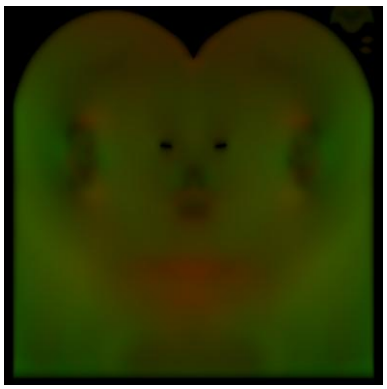

Figure 8: Blurred stretch map

9 Diffuse Color variation

One of the properties of diffusion profiles is that the color that is not absorbed will be the color of the material we want to render, however, this could not be applied to human skin as the color of it has many high-frequency changes, which is why we depend on the diffuse texture.

9.1 Post-Scatter

In this method, we will compute the irradiance maps as if they were white, then we will scatter the computed light, and finally, we will multiply the result by the diffuse texture.

By doing this, we will not blur high-frequency details, but we will not get color bleeding either. However, as the texture is usually made by a 3D detailed scan, color bleeding is already computed, so that would not be a flaw.

Figure 9: Mix value of 0

9.2 Pre-Scatter

The idea here is to compute the irradiance maps with the diffuse map. The problem here is that high-frequency details will be blurred.

Figure 10: Mix value of 1

9.3 Combination

An easy and physically accurate way is to mix the two previous methods. We will take into account the diffuse color when computing the irradiance map and in the final assembly. However, we will use a mix value to decide how it will affect each stage.

$$Before \rightarrow pow(diffuse, mix)$$
$$After \rightarrow pow(diffuse, 1 - mix)$$

10 Exact Energy Conservation

One of the most common things to do when building a rendering system is to separate the specular and diffuse terms and then combine them using an arbitrary constant for each of them; this works when the material we are rendering does not have lots of changes in frequency regarding color and curvature. This means that for skin, this method will not suit correctly, which is why in this implementation the diffuse term depends on the specular term in the following way.

$$kd = 1 - \int_{2\pi} f_r \cdot (x, \omega_0, L) \cdot (\omega_0 \cdot N) \cdot d\omega_0$$

As the previous integral is extremely heavy computationally speaking, we can precompute and store the values into a square texture just as we did with the Beckmann texture. We will access this texture with the angle of the normal and light vector, and with the roughness.

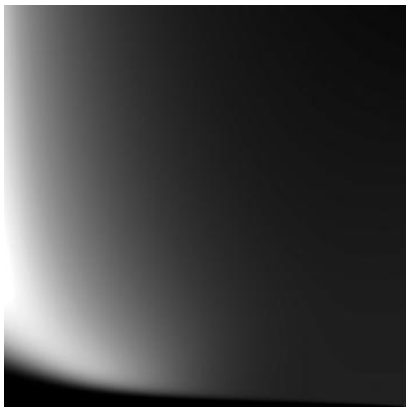

Figure 11: Attenuation Texture

11 Translucent Shadow Maps

We have already discussed how subsurface scattering and reflection are implemented, but previous sections do not explain how light works when it passes completely through a surface, as it happens with ears and nostrils. To explain this phenomenon, a modified version of Translucent Shadow Maps [Dachsbacher and Stamminger 2004] is used.

In the paper modification [d'Eon et al. 2007], when computing shadow maps, we will render the Z coordinate of the light facing surface and the UVs. However, I decided to store the Z coordinate in perspective space, the absolute value of the Z coordinate in the camera space (as the computations are done taking distances in world space), and the UVs. With this, when we are computing the irradiance maps we will be able to sample the TSM and check if we are rendering a point that is back facing or front facing the light, and if we are a back facing point we can store a thickness value (the distance between the points where the light enters and where the light exits the surface) in the alpha channel of the irradiance map.

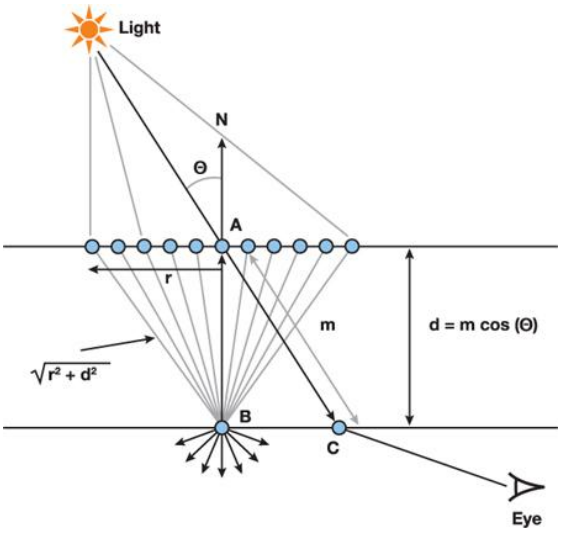

Figure 12: Region in shadow [d'Eon et al. 2007]

For point C, we need to add the scattered light of the neighborhood of point A. With TSM we will have access to the other side of the surface and to the distance to the light of it, and as we want to approximate the light scattered exiting at C, we will use the blurred textures. However, computing

this for point B is easier, and as we have the irradiance map already blurred, that scattering is computed already and will need to be added by a weighted sum into the final assembly, again, this is possible thanks to the properties of Gaussians and convolutions.

Finally, the depths used for the thickness are corrected by the angle formed by the normal at points C and A if they are opposite. Then we will make a lerp between the corrected thickness and the previously computed thickness. After that, as previously stated, we will store this value in the alpha channel of the irradiance texture to blur it to avoid artifacts in sections where high-frequency changes happen. Another important thing is to make the surface points facing the light have a high thickness value to avoid this computation effect.

12 Results

As we can see in the demo of my implementation and the images provided by the original paper, the results are pretty similar. Obviously, as the assets are not the same, the results are not exactly equal, but we can observe how the skin gets its soft appearance while also having a realistic specular reflection and pretty accurate translucency in thin regions.

There are some artifacts that happen when using the energy conservation texture, but they are quite subtle. Also, if we use the Beckmann texture, some specular spots appear black, so I decided to use the Beckmann function to avoid them.

To see more results, go to the end of this paper.

13 Possible improvements

One of the things that will be of high priority to work on is performance. As it is a Vulkan renderer done from scratch without previous Vulkan experience, it was built with the knowledge that I learned as I was implementing the paper, so many things of the framework were thought to be reworked once the implementation stages were more advanced.

Another thing that would be a nice addition would be the possibility of having multiple lights and environmental lights. The paper explains that this could be done easily without needing to add extra blur passes. We could do this easily by precomputing the light map for every light into the same texture and by only taking the minimum thickness values for each point of the surface.

Finally, something that could be added is the option to have more assets loaded for variety, and also fixing the problem with the seams of the assets. However, this is something that will likely require an artist to fix easily.

14 Vulkan Experience

My journey with Vulkan has been full of ups and downs, but in general, I can say that I learned a lot and that I have learned the use of modern graphics APIs.

The start of the project was very rough. There is almost no information on the internet about some specific things about Vulkan, however, there was a nice tutorial that acted as a starting point and as a guide through the project. Also, coming from OpenGL, which is a much more high-level language, made me change my mind about how new graphics APIs work.

One of the things that I appreciate most when writing code in Vulkan is that is extremely specific, which is good because it almost allows you to do everything you want, but that also means that requires you to know exactly what you are doing to avoid getting errors, and in addition to that, it means that a single thing could need lots of lines of code to work correctly. However, Vulkan has a nice system of validation layers, which allow you to know if you are doing something wrong in an easy way, being clearer than OpenGL.

And although the validation layers help, sometimes you will get errors that are not solved easily and that require you to search on the internet. This would not be a problem with OpenGL, as it is an old API, however, Vulkan is fairly new and some problems do not have a solution in a blog or in the official documentation, which means you will have to fight and debug your code slowly.

15 Development problems

There were various problems found when developing this project, but the most important ones are the following.

First of all, I had to learn a new API from zero which was not like OpenGL, which was the API that I previously used. Because of that, the development of the paper itself was slowed down due to having to refactor parts of the framework or having problems that were not solved easily, for example, synchronization problems.

Then, as the project was done with assets found on the internet, it was impossible to get the same result as in the original implementation, so most of the time, after getting something done, I had to keep implementing the rest of the techniques without being sure if previous implementations were exactly correct. This was a headache when implementing TSM.

Finally, as this was an independent study and it depended completely on my organization, sometimes I could not dedicate all the time I wanted to the project because I had to do other things related to the university.

16 Conclusions

After doing this project, I was able to learn a new API, new techniques that explained the theory of how light interacts with some materials, and how to organize myself better.

This paper is a very complete implementation that talks a lot about other people's work and allows you to learn more while you are implementing it. The results that it gives are suitable for Real-Time Rendering if we reduce the resolution of the textures used and the space the characters could take up on the screen. Also, the results are very accurate compared to some other offline rendering techniques.

17 References

d'Eon, Eugene, David Luebke, and Eric Enderton. 2007. "Efficient Rendering of Human Skin." In *Rendering Techniques 2007 (Proceedings of the Eurographics Symposium on Rendering)*, Grenoble, France.

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

Donner, Craig, and Henrik Wann Jensen. 2005. "Light Diffusion in Multi-Layered Translucent Materials." In ACM

Transactions on Graphics (Proceedings of SIGGRAPH 2005) 24(3).

Donner, Craig, and Henrik Wann Jensen. 2006. "A Spectral BSSRDF for Shading Human Skin." In Rendering Techniques (Proceedings of the Eurographics Symposium on Rendering 2006), pp. 409–417.

Krishnaswamy, Aravind, and Gladimir V. G. Baranoski. 2004. "A Biophysically-Based Spectral Model of Light Interaction with Human Skin." In Computer Graphics Forum (Proceedings of Eurographics 2004) 23(3).

Kelemen, Csaba, and László Szirmay-Kalos. 2001. "A Microfacet Based Coupled Specular-Matte BRDF Model with Importance Sampling." Presentation at Euro-graphics 2001.

Dachsbacher, Carsten, and Marc Stamminger. 2004. "Translucent Shadow Maps." In *Proceedings of the 13th Eurographics Workshop on Rendering*, pp. 197–201.

Cook, Robert, and Kenneth Torrance . 1982. "A Reflectance Model for Computer Graphics." Pixar.

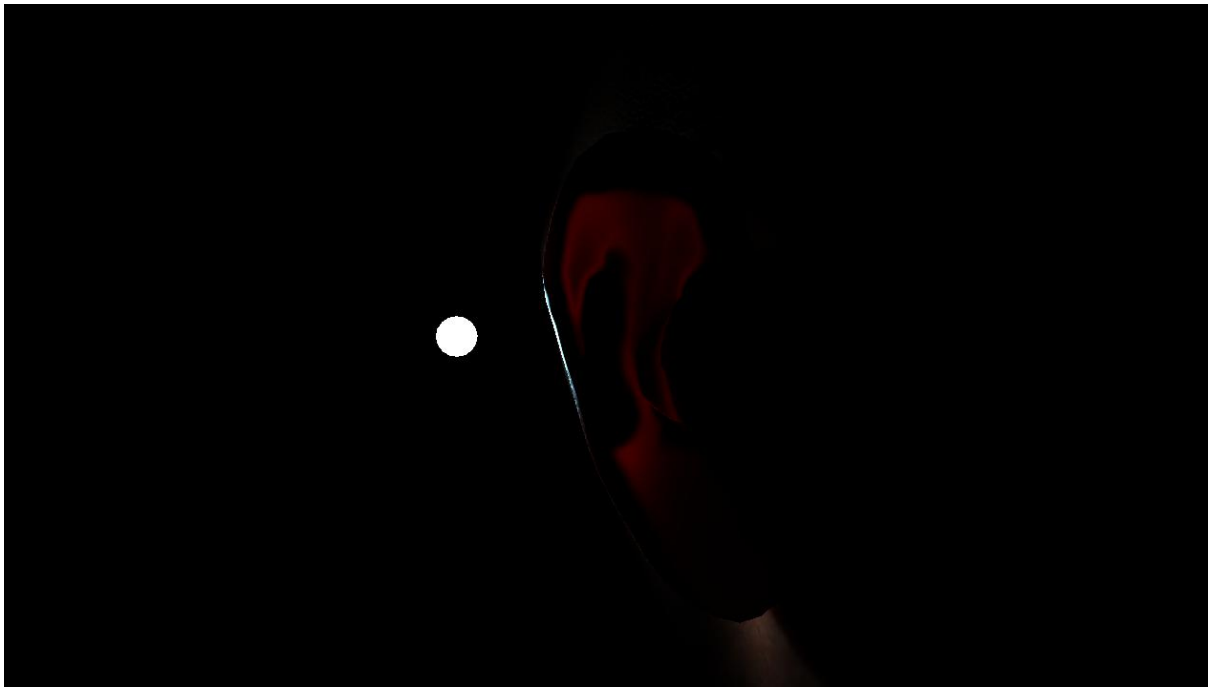

Figure 13: Ear with this technique

Figure 14: Nose with this technique

Figure 15: Lighting with this technique

Figure 15: Lighting with Phong (for comparison)