Realtime Subsurface Scattering

P. Schönberger

RheinMain University of Applied Sciences, Wiesbaden, Germany

Abstract

Simulating subsurface scattering, the process of light entering a material and exiting it at a different point, is crucial to rendering certain translucent materials realistically, but is very computationally expensive if done in a physically correct way. To enable realtime rendering of such materials, several methods have been devised that ease the process by approximating this effect. These drastically reduce the effort involved in simulating subsurface scattering, by making some basic assumptions about the behaviour of light and it's interactions with objects.

1. Introduction

When light shines on a surface, most of it is reflected and some is absorbed. But depending on the material, some light rays might enter the surface, bounce around inside the material and leave it again. This is called subsurface scattering (SSS). Because these light rays often leave the material at a different point than they entered, this alters the appearance of the material. Good examples of this are milk or human skin. When trying to render these kinds of materials digitally, not paying attention to this effect will make them look artificial because they are too smooth. So SSS has to be simulated in order to be able to render a human realistically.

1.1. Approximating Subsurface Scattering

If you want to simulate the effect of SSS in a physically correct way, you have to simulate every individual light ray and calculate its path through a material. This will yield a realistic result, but it is computationally expensive and currently not feasible to do in real time [Pet19]. So the effect has to be approximated.

1.2. Use Case

The motivation behind this project was to enable a more realistic rendering of teeth. Rendering teeth solely based on color and shape makes them look like plastic, because they consist of several layers with differing translucency, giving them a very specific appearance.

2. Approximating Subsurface Scattering

Instead of simulating every single light ray to determine it's path through a material and get an accurate result, we can make some assumptions that simplify the problem without affecting the result too much. First, the distribution of light in a highly scattering material is effectively isotropic [JMLH01]. This means, that we can apply a distribution function to the point where light enters the material and get a result that is almost identical to an exact simulation.

Depending on the approximation method, other assumptions will be made. Two such methods will be presented, Screen Space SSS and Texture Space SSS. Both have been implemented and will be reviewed in greater detail in 3.

2.1. Screen Space SSS

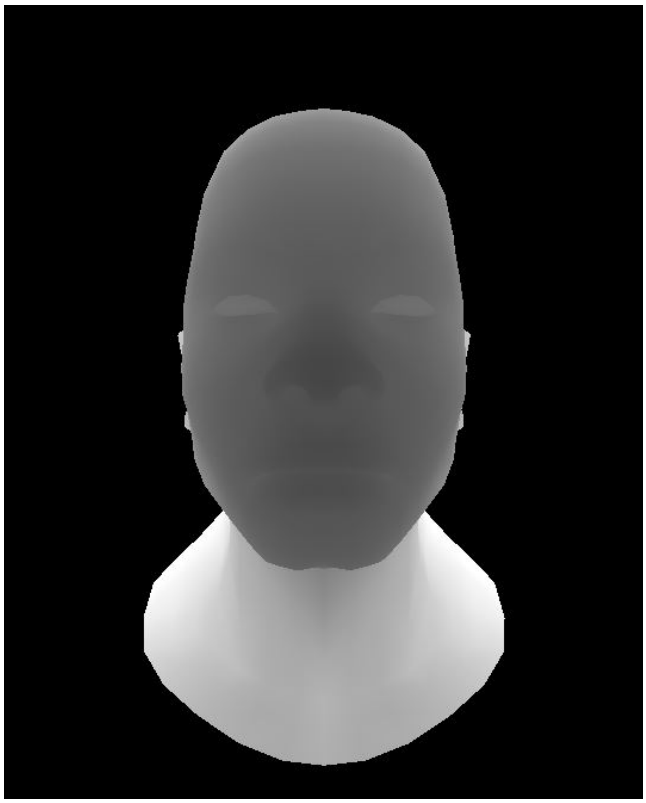
Screen Space SSS (SSSSS) is a comparatively fast approximation for SSS. The basic idea is that two points that are close to each other on screen are usually also close to each other in world coordinates. Of course this isn't always true, but in the general case it works well enough to produce convincing results.

This method uses multiple render targets (MRTs) to produce a shadow map and later use that to calculate irradiance and translucency. A first render pass is done from the position of the light source looking in the direction of the object to be illuminated, with the color encoding the distance to the light, this is the shadow map (see Figure 1).

In a second render call, the model is rendered regularly, the way it would be without any SSS. For each fragment, the corresponding texture coordinates in the shadow map are then calculated by unprojecting the fragment's world coordinates into the light's local space. The value sampled from the shadow map is then multiplied with the previously calculated color and the dot product of the normal and the angle at which the light hit the surface to produce irradiance.

By sampling from the shadow map with a Gaussian filter, light scattering through the material is approximated. Since two fragments that are close to each other usually correspond to two points that are close to each other on the object, the two dimensional blur can emulate local light transmission.

Figure 1: *Shadow map rendered from light's position*



Since this only works for points that are close to each other on screen, translucency has to be applied separately. The shadow map is used for this as well. Using the shadow map we can find the world coordinates of the fragment that was used to produce the shadow map. By calculating the distance from that fragment to the current one, we get the objects thickness at that point and can calculate translucency. This of course assumes, that the object is convex, because any spaces present between the two points will not be considered, even though they would stop light propagation.

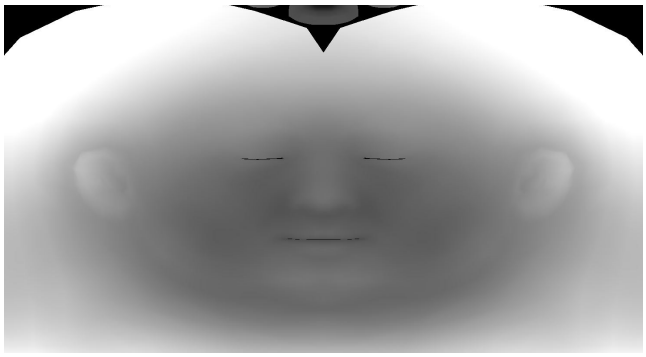
2.2. Texture Space SSS

The underlying simplification for Texture Space Subsurface Scattering (TSSSS) is similar to the one made for SSSSS. If two points' UV coordinates are close to each other, the points are close to each other on the object. This is of course broken by UV seams, but again, it works well enough in the general case to produce satisfying results.

We are using MRTs again, to produce an irradiance and a shadow map. The irradiance map is produced by laying out the model according to it's UV coordinates in a 2D plane. Irradiance is calculated again by multiplying the distance to the light source by the dot product of normal and light entry angle. The resulting texture (see Figure 2) maps UV coordinates to their irradiance.

The shadow map is produced similarly to the way previously

Figure 2: *Irradiance map for model laid out according to it's UV coordinates*



described, but it stores the fragment's UV coordinates in addition to the light distance.

When rendering the model, the current UV coordinates are simply used to sample the irradiance (i) and multiply it with the previously obtained color (c) as $\sqrt{i * c}$ to get a normalized result. Translucency is introduced analogously to how it was calculated for SSSSS, the only difference is, that irradiance is sampled from the irradiance map using the previously stored UV coordinates.

3. Implementation

Both Screen Space and Texture Space Subsurface Scattering have been implemented as GLSL shaders in OpenGL. Both support scattering light using a Gaussian kernel and adding translucency according to a shadow map. The materials interaction with light can be influenced by changing the blur kernel and varying the term with which translucency is calculated.

3.1. Application

The encompassing application is written in C++ and uses SFML to open an OpenGL capable window. Models are loaded using Assimp and there is a menu that allows manipulation of variables such as light position or light transmission in the material at runtime. An orbiting and a free camera are available to view the model.

3.2. Screen Space SSS

Two framebuffers are used to store a shadow map and a regularly lit version of the model. The shadow map is produced by calculating a perspective projection from the light source's position towards the object. The distance is calculated by passing each fragment's world coordinates from the vertex to the fragment shader. There, the distance between the light's and the fragment's position is calculated and stored as a brightness value (see Figure 1).

In the second render pass the model is rendered to a second framebuffer from the camera's perspective and is lit using simple Phong shading. In the fragment shader, the fragment's texture coordinates are looked up in the shadow map. The resulting distance is used to calculate the world coordinates of the fragment that was

X	Y	R	G	B
0.0000	0.0000	0.2204	0.4870	0.6350
1.6339	0.0367	0.0763	0.0644	0.0390
0.1778	1.7175	0.1165	0.1032	0.0649
-0.1949	0.0910	0.0648	0.0863	0.0622
-0.2397	-0.2202	0.1317	0.1516	0.1036
-0.0035	-0.1182	0.0256	0.0427	0.0330
1.3201	-0.1815	0.0485	0.0647	0.0461
5.9706	0.2533	0.0480	0.0030	0.0004
-1.0892	4.9583	0.0488	0.0054	0.0012
-4.0154	4.1566	0.0513	0.0060	0.0014
-4.0630	-4.1101	0.0614	0.0091	0.0025
-0.6386	-6.2976	0.0309	0.0028	0.0006
2.5423	-3.2459	0.0735	0.0232	0.0097

Table 1: *A Gaussian kernel to approximate local scattering of light. X/Y sample locations are given with weights for each RGB channel.*

originally present in the shadow map. This is done in two steps. First, find the vector that points from the light's position towards the point by subtracting the light's position from the points world coordinates. Then, add the normalized vector, multiplied by the obtained distance, to the light's position. Now we need to distinguish two cases. If the calculated point, the fragment's world coordinates, is identical to the fragment, that means the fragment is directly lit. In that case no translucency has to be applied. If they are not identical, the calculated point is the backside of the lit fragment. In this case, the fragment's color is multiplied by $x^{y/d^{0.6}}$, with x and y being constants that determine the rate at which light scatters inside the material. Tweaking this constant produces different levels of translucency, as can be seen in Figure 4, where the model looks like it is made out of glass or wax. d is the distance between fragment and backside.

With this we have a lit model with simulated translucency, but no scattering of light between points that are close to each other on the surface. For this, the image rendered to the second framebuffer is displayed on a rectangle mesh exactly the size of the screen. In the fragment shader the image is sampled using a Gaussian kernel taken from [HBH09] to emulate local light scattering (see Table **??**).

This produces a glow around the outline of the model, which can be avoided by using a stencil buffer to limit where the fragment shader is actually applied on the screen mesh. The result can be seen in Figure 3.

3.3. Texture Space SSS

For Texture Space Subsurface Scattering we also do three render passes. Two for producing an irradiance and a shadow map and a last one to render the model with light scattering and translucency provided by the previous renders.

While the shadow map is similar to how it was made in screen space, the irradiance map was not needed in screen space. It is produced by positioning the model in the XY-plane using it's UV coordinates as world coordinates. The vertex shader can be seen in

Figure 3: *A model of a head rendered without (left) and with (right) Screen Space Subsurface Scattering*

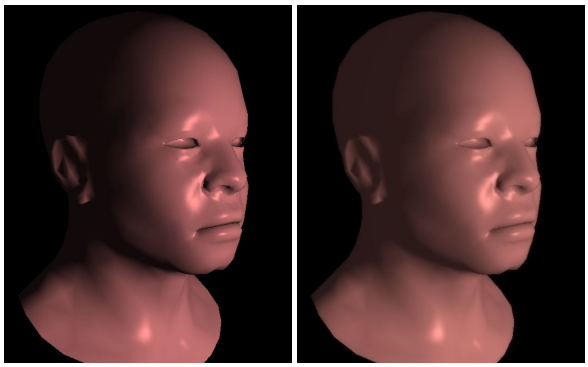
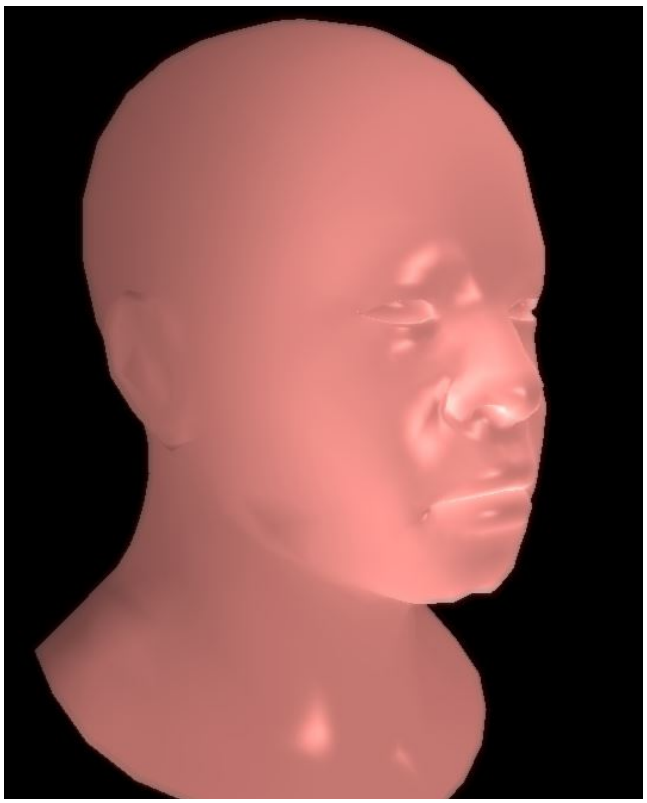


Figure 4: *Altered translucency term producing differenct material*



Listing 1. It maps the UV coordinates from their original range of 0 ... 1 to -1 ... 1 so they match OpenGL's normalized device coordinates.

Listing 1: *Vertex shader for the shadow map*

```
void main()
{
  gl_Position = vec4(uv * 2.0 - 1.0, 0.0, 1.0);
  FragPos = vec3(model * vec4(pos, 1.0));
  Normal = normal;
}
```

The same Gaussian kernel from [HBH09] is used to sample irradiance and translucency is also calculated with the same term. The result can be seen in Figure 5.

Figure 5: *Texture Space Subsurface Scattering*

4. Conclusion

Both methods are largely implemented as GLSL shaders and are easily able to run in realtime. Apart from the obvious issue with locality not always transferring from screen/texture space to 3D, both methods produce results that look more life-like than plain Phong shading. The texture space method has the added problem of failing at UV seams and suffering if the UV map is stretched and both suffer from the fact, that a shadow map has to be rendered for each light source, but all in all they enable reasonable simulation of light scattering without too much of a performance penalty.

References

[HBH09] HABLE J., BORSHUKOV G., HEJL J.: Fast skin shading. *Shader X7* (2009), 161–173.

[JMLH01] JENSEN H. W., MARSCHNER S. R., LEVOY M., HANRAHAN P.: A practical model for subsurface light transport. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 2001), SIGGRAPH '01, Association for Computing Machinery, p. 511–518. URL: https://doi.org/10.1145/383259.383319, doi:10.1145/383259.383319.

[Pet19] PETINEO M.: An introduction to real-time subsurface scattering. https://therealmjp.github.io/posts/sss-intro/, October 2019.