Spring 2013

Separable Subsurface Scattering: An OpenGL Implementation

Michael Tetzlaff
Bethel University

Separable Subsurface Scattering: An OpenGL Implementation

Michael Tetzlaff

May 9, 2013

Abstract

The light reflection of many materials, including skin, milk, and marble, are not dominated by direct reflection, but rather by the subsurface scattering of light [1]. Most real-time computer graphics techniques, most notably Phong lighting, do not account for subsurface scattering of the diffuse reflectance term. Recently, however, Jorge Jiminez has produced an efficient technique involving only two screen-space blur passes which can achieve the effect of subsurface scattering in real time[3]. Jiminez' implementation was in Microsoft's DirectX, so this project is an attempt to implement his technique in OpenGL, which has free open-source implementations.

Previous Implementations of Subsurface Scattering

Subsurface scattering is not a new technique in computer graphics. Henrik Wann Jensen was the first to develop the mathematics behind subsurface scattering and create a model for how translucent materials could be efficiently rendered[1]. D'Eon et al. were recently able to achieve real-time subsurface scattering by doing multiple blur passes in texture space[2]. However, Jorge Jiminez' recent work has shown that this is not necessary to achieve a realistic effect[3]. Jiminez has developed a method for subsurface scattering that requires only two blur passes in screen space. The fact that the rendering can be done in screen space means that the algorithm scales much better, due to the fact that the number of passes does not depend linearly on the number of objects in the scene. While Jiminez's implementation used Microsoft's DirectX, this project is an attempt to implement his subsurface scattering technique in OpenGL.

Transmittance

An essential part of the subsurface scattering effect is that some light is transmitted through thin, otherwise opaque objects. Generally, transmittance can be expressed as a function of thickness: $T(d)$. d'Eon et al. have shown that this function can be approximated as a linear combination of Gaussians with different variances [2]. The weights and variances can be determined by experimentally measuring the transmittance function from physical materials, and fitting the data to an arbitrary linear combination of N Gaussians. On the other hand, someone with an artistic eye could also create this function from scratch by tweaking the parameters by hand until the material looks realistic. The transmittance function is wavelength dependent, meaning that the red, green, and blue channels may each have a different transmittance function. For skin, the red channel is transmitted the furthest, and the blue channel is transmitted the least. On the face model, this effect is most noticeable behind the ears and the nose, as shown in Figures 1 & 2.

Transmittance can be rendered in real-time in a manner similar to shadow mapping. Just as in shadow mapping, the scene is rendered from the light source's perspective, and the depth of the closest point to the light stored for each pixel [4]. Then, in the fragment shader for the main rendering pass, the fragment's position is transformed into "light coordinates" using the same model, view, and projection matrices from the light perspective pass. However, whereas with shadow mapping all that matters is whether the fragment is in front of or behind the closest point to the light, for transmittance, the distance between the fragment and the closest point must be found. Because of this a critical difference is necessary for generating a depth texture for transmittance: the depths must be stored linearly, as opposed to the exponential format usually used for the depth buffer in the scene. For efficiency, either the linear depth map for transmittance can

Figure 1: Nose without subsurface transmittance (left) and with subsurface transmittance (right)

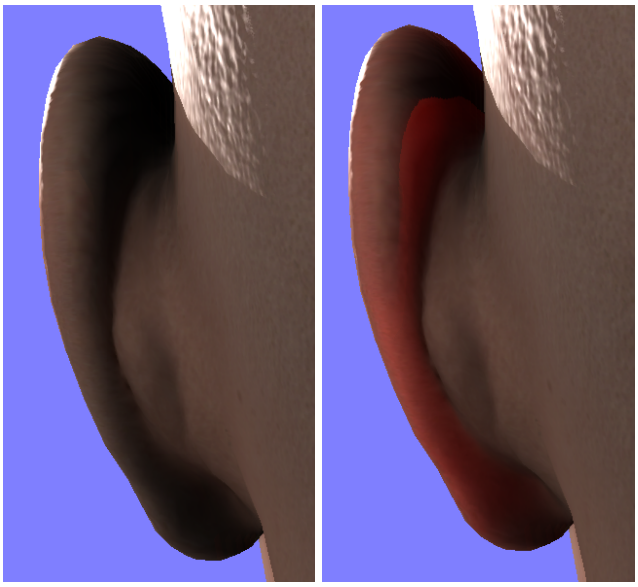

Figure 2: Ear without subsurface transmittance (left) and with subsurface transmittance (right)

be calculated in the same pass as the exponential depth map for shadow mapping, or the shadow mapping function can be adapted to use a linear depth map. Another tricky implementation detail is that the scale is very touchy, because if the "critical depth" for noticeable transmittance is too large, the effect will not be seen, while if the "critical depth" is too small, it may result in everything looking translucent, even very thick shapes such as the man's skull (shown in Figure 5).

Blurring of Diffuse Term

Transmittance accounts for light which enters the surface of a material, and exits on the "back side" from the light source's perspective. However it does not account for light which enters the surface, is scattered, and then is reflected back towards the original light source. The diffuse term of Phong lighting covers light which is scattered off the surface itself, but not scattering below the surface. The easiest way to account for this form of subsurface scattering is to add blur passes after the diffuse term of Phong lighting scene has been rendered. Two blur passes, a vertical pass and a horizontal pass, are all that is necessary to give a realistic impression of subsurface scattering, and these passes can be done in screen space. Only the diffuse term should be blurred, so the specular term for the scene should be rendered in a separate texture, and added on top of the scene after the blur passes are complete.

The blurring technique can be thought of as replacing each pixel with a weighted sum of several pixels in the surrounding area. These weights can be determined from the same transmittance function used for translucency: in this case $T(d)$ represents the weight that a particular pixel has in the blur function for a pixel that is a distance d away. For efficiency, the transmittance function can be stored in a "kernel" texture which contains several well-chosen distances at which to sample the scene, along with the value of the transmittance function at those points. The blur shader then looks up the pixels at the distances stored in the kernel, multiplies them by the weights, and sums them together to find the final color for each pixel. Once again, the scale of the effect is essential; blur distances that are too small will not be noticeable, but blur distances that are too big (greater than the number of pixels in the kernel) will reveal the discrete lookups and ruin the effect.

A comparison of the rendering with and without this blur pass can be seen in Figure 3.

Figure 3: Skin without blur pass(left) and with blur pass (right)

This technique has one caveat - it does not take into account the fact that many surfaces are not parallel to the projection plane, and there may be drastic changes in depth along a boundary between a foreground object and a background object. The problem with this is that an object in the foreground may be blurred with an object in the background, which should not happen physically. Jiminez has a simple solution that seems to mostly resolve the issue: he looks up the depth values of the two points which may be blurred together, and puts the difference into a function which linearly interpolates back to the original color as the difference in depth increases. At some maximum depth, this function will be zero, and there will be no blurring across the boundary, which is the desired behavior between a foreground and a background object. This maximum depth is another parameter which would need to be tweaked for a particular application. This

solution is not particularly elegant, but it seems to be a decent workaround given the performance advantage that Jiminez' implementation has. A comparison between a rendering without any depth consideration and with Jiminez' linear interpolation solution is shown in Figure 4.

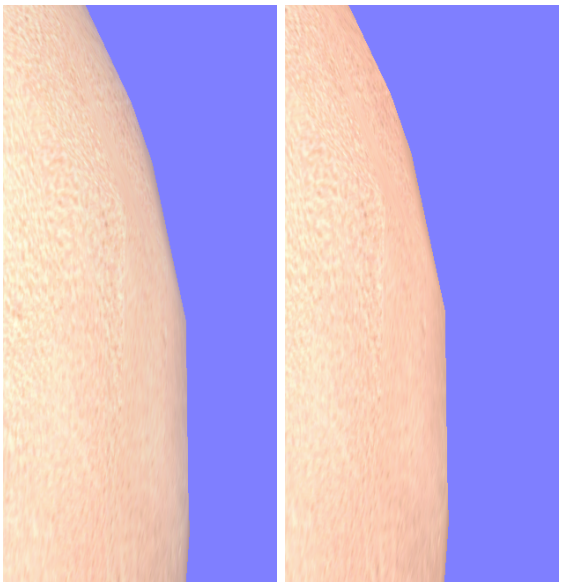

Figure 4: Edge of head without depth consideration in blur pass (left) and with depth consideration (right). Notice the undesirable blurring with the background in the first case.

One factor that Jiminez' algorithm does not account for is the angle between the surface being rendered and the projection plane - his algorithm assumes that all faces are essentially parallel to the projection plane. Visually, this does not seem to really cause any major flaws in rendering the face model, but a model could probably be contrived where this would be a problem. A potential solution could be to store a texture of normal vectors in screen space, so that a simple texture lookup could determine the angle of the face and adjust the scale of the blur effect in each direction accordingly.

Results

With an inexpensive three-year old laptop graphics card (NVidia GeForce 9400M) the OpenGL implementation ran at about 7fps - not very good, but taking into account Moore's Law that processor speeds tend to double every two years, within the next year or so, an average new GPU should be able to easily perform this technique at over 30fps. For comparison, Jiminez' implementation ran at about 9fps, which is probably because he may have made some tweaks to optimize the program running on DirectX. Because the technique is done in screen space, it should also scale well in a scene with many objects - the only variables affecting the performance are the screen resolution and the number of samples to use in each blur pass.

A comparison of the entire head model, with and without subsurface scattering, is shown in Figure 5.

Conclusion

Realistic computer rendering of human skin in real-time seems to finally be within reach. As GPU speeds continue to increase over the next few years, it should be possible to easily perform screen-space subsurface scattering using Jiminez' technique for many objects in a single scene without seeing a major frame-rate hit. Along with Jiminez' DirectX implementation, this OpenGL implementation shows that it is flexible enough to be implemented on many different platforms. The technique is relatively simple and elegant, and should be used to complement and complete any implementation of Phong lighting which needs to realistically render translucent materials.

References

[1] Jensen, Henrik Wann, et al. “A practical model for subsurface light transport.” Proceedings of the 28th annual conference on Computer graphics and interactive techniques. ACM, 2001.

[2] d’Eon, Eugene, David Luebke, and Eric Enderton. “Efficient rendering of human skin.” Proceedings of the 18th Eurographics conference on Rendering Techniques. Eurographics Association, 2007.

[3] Jiminez, Jorge. “separable-sss (source code).” https://github.com/iryoku/separable-sss

[4] Williams, Lance. ”Casting curved shadows on curved surfaces.” ACM Siggraph Computer Graphics. Vol. 12. No. 3. ACM, 1978.

Figure 5: Full model without subsurface scattering (left) and with subsurface scattering (right)