CEIG'08, Barcelona, Sept. 3–5 (2008)
L. Matey and J. C. Torres (Editors)

Faster rendering of human skin

Jorge Jimenez and Diego Gutierrez

Universidad de Zaragoza, Spain

Abstract

Rendering realistic human skin is not a trivial task. It means dealing with light diffusion in a multi-layered material, where multiple subsurface scattering events take place. Great care must be taken when simulating its appearance, to avoid an unnatural, waxy look. Recent works in the field of computer graphics have provided us with the ability to accurately render human skin, both off-line and in real time. The latter takes advantage of modern graphics hardware, defining light diffusion in texture space. In this paper we leverage this framework and optimize the irradiance map calculations, which simulate light diffusion in skin. We present three simple yet effective improvements implemented on top of the state-of-the-art, real-time rendering algorithm published by d'Eon et al. We achieve maximum speed-ups in excess of 2.7*x using the same hardware configuration. Our implementation scales well, and is particularly efficient in multiple-character scenarios. This should be specially useful for real-time realistic human skin rendering for crowds.*

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Three-Dimensional Graphics and Realism

1. Introduction

Translucency is a very common material property: paper, tree leaves, soap, a candle, fruit... these are all common objects which present a certain degree of subsurface scattering (see Figure 1), and thus pose a challenging problem in the field of computer graphics; light transport *within* the objects' surface must be correctly simulated in order to accurately capture their appearance.

Skin is a highly translucent material, made up of multiple translucent layers. We refer the reader to the excellent review by Igarashi and colleagues [INN05] for a complete breakdown of its components. Each layer scatters light according to its specific composition, providing the characteristic reddish look. The human visual system is very well tuned to certain aspects of an object's appearance, and human skin seems to be one of them: if its shading is a bit off, it will look waxy, dead.

Subsurface scattering (SSS) is usually described in terms of the Bidirectional Scattering Surface Reflectance Distribution Function (BSSRDF) [NRH*77]. The first attempts at simulating it were based on the assumption that light scatters at a single point on the surface, adding a diffuse term to account for the overall appearance of SSS effects. Hanrahan and Krueger [HK99] only took into account single scattering, whilst Stam [Sta01] extended the idea to simulate multiple scattering. Jensen and co-workers provided a huge step forward and made SSS practical, publishing techniques that were rapidly adopted by the movie industry [JMLH01, JB02]. Based on a dipole approximation of light diffusion, they were able to capture the subtle softness that translucency adds to the appearance of skin.

Figure 1: Tree leaves and candle showing high translucency properties.

Recently, Donner and Jensen extended their dipole-based model to multiple dipoles, which can also capture the effects of discontinuities at the frontiers of multi-layered ma-

© The Eurographics Association 2008.

terials [DJ05]. They show results with a broad range of objects, and their skin simulations look particularly impressive. Unfortunately, the model relies on a high number of parameters that need to be measured in advance, which makes its direct application to rendering a bit cumbersome. The same authors partially overcome this issue in [DJ06], presenting a physically-based spectral shading model for rendering human skin which requires only four parameters. These parameters specify the amount of melanin, hemoglobin, and the oiliness of the skin, which suffice to provide spectacular renderings.

However, the computation time needed for any of these models is still in the order of several seconds (or even minutes) per frame, which rules out any real-time application. There is a good deal of research aiming to provide SSS approximations which work sufficiently well, sacrificing physical accuracy in exchange of speed. Most of them impose the burden of heavy precomputation times, though [HV04, WTL05, WWD*05]. One of the most popular techniques, Precomputed Radiance Transfer [SKS02] imposes additional restrictions, such as fixed geometry that makes animation of translucent meshes impracticable. Modern graphics hardware has also been used: Borshukov and Lewis [BL03] use 2D diffuse irradiance textures, with SSS simulated by a Gaussian function with a customizable kernel. The technique maps naturally onto GPUs, but it still fails to capture the most complex subtleties of multiple scattering within materials. Dachsbacher and Stamminger [SD03] introduce the concept of translucent shadow maps, a modified version of shadow maps extended with irradiance and surface normal information. In concurrent work, Mertens et al. [MKB*05] presented their own algorithm for local subsurface scattering, which is the same in spirit as [SD03].

The work by d'Eon and colleagues at nVIDIA Corporation [DLE07, DL07] simplifies the models of Donner and Jensen [DJ05, DJ06] combining them with the idea of diffusion in texture space [BL03, SD03]. They approximate the multiple dipole scheme with a sum of Gaussian functions, obtaining separable multi-layer diffusion profiles which are combined in a final render pass. Visual inspection of their results match the off-line renderings of Donner and Jensen, but they are achieved at real-time frame rates. In this work we build on [DLE07, DL07] and propose three simple enhancements to the technique, which have been tested on a nVIDIA GeForce 8800. These enhancements allow us to achieve maximum speed-up factors in excess of 2.7x using the same hardware configuration.

In the following section we provide an overview of the work by d'Eon and colleagues, before introducing our three optimization techniques. We then provide results, with conclusions and future work at the end.

2. Efficient Rendering of Human Skin

The work by d'Eon and colleagues [DLE07] shows how texture-space diffusion [BL03] can be combined with translucent shadow maps [SD03] to create a very efficient, real-time rendering algorithm for multi-layered materials with strong subsurface scattering. They apply it to human skin, achieving impressive results. Given that the optimization techniques we present in this paper build on that work, we first contextualize it in this section, for the sake of clarity.

The dipole model: To allow for an efficient simulation of light scattering in highly scattering materials, Jensen et al. [JMLH01] approximate the outgoing radiance L_0 in translucent materials for a dipole diffusion approximation. Thus, the costly Bidirectional Scattering Surface Reflectance Distribution Function (BSSRDF) becomes:

$$S_d(x_i,\vec{\omega}_i;x_o,\vec{\omega}_o) = \frac{1}{\pi}F_t(x_i,\vec{\omega}_i)R(\|x_i-x_o\|_2)F_t(x_o,\vec{\omega}_o) \quad (1)$$

Where x_i and $\vec{\omega}_i$ are the position and angle of the incident light, x_o and $\vec{\omega}_o$ are the position and angle of the radiated light, F_t is the Fresnel transmittance and R is the diffusion profile of the material. The model is further simplified in subsequent work by Jensen and Buhler [JB02].

The multipole model: Donner and Jensen [DJ05] extend the dipole model to a multipole approximation, defined as a sum of dipoles. The model works well for thin slabs (thus removing the semi-infinite geometry restriction in [JMLH01, JB02]), and accounts for surface roughness and refraction effects at the boundaries. For each pair of slabs, they analyze the convolution of their reflectance and transmittance profiles in frequency space (thus performing faster convolutions instead). They show how the multipole approach provides a better fit with ground-truth Monte Carlo simulations than the simpler dipole approximation. Fresnel reflectance accounts for differences in the indices of refraction for the boundaries of a slab. In the presence of rough surfaces, the Torrance-Sparrow BRDF model [TS67] is used instead. Coupled with a Monte Carlo ray tracer, the authors show a wide range of simulated materials, such as paper, jade, marble or human skin. Unfortunately, the model relies on involved precomputations, in the order of a few seconds per frame, which rules out real-time applications of the method.

The Gaussian approximation: The recent work by d'Eon and colleagues [DLE07] is based on one key observation: the reflectance and transmission profiles that the multipole model predicts can be approximated by a weighted sum of Gaussians. Let $R(r)$ be a radial diffusion profile, then the error term:

$$\int_0^\infty r\left(R(r)-\sum_{i=1}^{k} w_i G(v_i,r)\right)^2 dr \quad (2)$$

© The Eurographics Association 2008.

must be minimized, where w_i is the weighting factor and v_i represents variance. Both parameters are user-defined, along with the number of curves k, which is usually set between two and six. The authors report errors between 1.52 and 0.0793 percent for the materials measured in [JMLH01]. This observation allows for a much faster computations, given that 2D convolutions can now be separated into two cheaper, 1D convolutions in x and y respectively. Additionally, obtaining the diffusion profile for two slabs put together does not require a frequency-domain operation, since the radial convolution of two Gaussians $G(v_1)$ and $G(v_2)$ is simply $G(v_1+v_2)$. This allows the fact that the long series of convolutions and additions computed in frequency domain by [DJ05] can now be efficiently approximated as polynomial multiplications and additions over Gaussians.

Texture-space diffusion: The above mentioned desirable properties of the diffusion representation allow the sum of Gaussians to be separated into a hierarchy of irradiance diffusion maps. These are rasterized into an off-screen texture, using a fragment shader for lighting computations and a vertex shader for mesh-to-texture mapping. The net result is a series of progressively smaller irradiance maps which when combined (weighted sum) closely match the non-separable global diffusion profile by Donner and Jensen [DJ05].

Figure 2 shows a scheme of the necessary steps involved in the nVIDIA shader. Our optimization techniques, explained in the next sections, are applied during the two most expensive steps: the irradiance maps calculation and the subsequent convolutions. Together with the cheaper shadow maps and sum of convolutions steps, these four steps depend on the number of objects being rendered and/or light sources (note that only one head is displayed in [DLE07], while our approach takes multiple objects into account). The final bloom pass, not optimized with our algorithms, is performed on the final image and thus its computation time does not increase as the number of objects grows.

3. First Optimization: Culled Irradiance Map

For sufficiently large textures, more than fifty percent of the rendering time is spent on obtaining the irradiance maps and convolving them with the different Gaussian functions. We base our first optimization on the observation that these textures are obtained for the whole model, regardless of the current viewpoint for each frame. We leverage the fact that of course not all the geometry will be seen at the same time, and thus we can take advantage of backface culling techniques. For each pixel in the irradiance maps we can first check whether it belongs to a point in the model which is visible in the current frame, since the surface normal is known at each point, and simply discard those pixels belonging to occluded parts of the geometry. Visibility information is stored in the alpha channel (0.0 means occluded, 1.0 means visible). This simplifies the computation of the maps and subsequent convolutions, given the reduced number of pixels containing meaningful irradiance values.

During convolution, the Gaussian functions should not be applied beyond the boundaries of the visibility map stored in the alpha channel. This would extend the convolution kernel to pixels where no irradiance information has been stored, thus potentially producing visible artifacts. To circumvent this, we modify the backface culling algorithm, extending it beyond the standard divergence of 90°. We thus store an occlusion value if the angle between the surface normal and the view vector is greater than 105°, which produces good results. Figure 3 shows the difference between the irradiance map in [DLE07] and our approach, for a given viewpoint. Occluded pixels have been highlighted in blue for visualization purposes (the red map belongs to the third optimization, introduced later in the paper).

We note that using vertex normals for backface culling can be a potential source of error: polygons conforming the apparent profile of the object are likely to have both visible and occluded vertices at the same time, and thus the algorithm would discard visible pixels. However, our experience shows that extending the culling angle to 105° greatly avoids any visible errors, while still providing noticeable speed-ups. A second possibility to circumvent this is to use attribute variables to store the normal of each polygon in the vertex shader.

Implementation issues: The following code illustrates the simple implementation of this optimization on the GPU. First we show the vertex shader (Listing 1) and pixel shader (Listing 2) for the irradiance map, followed by the convolution pixel shader (Listing 3). Visibility is computed on the vertex shader of the irradiance map to take advantage of automatic hardware pixel interpolation.

Listing 1: Irradiance vertex shader

```
void main() {
  vec4 eye = vec4(0.0, 0.0, 1.0, 1.0);
  vec4 n = gl_NormalMatrix * gl_Normal;
  n = normalize(n);
  eyedot = dot(eye, n);
  // ... transform vertex, etc.
}
```

Listing 2: Irradiance pixel shader

```
void main() {
  if (eyedot >= -0.2588) {
    // cos(105°)=-0.2588
    // ... calculate pixel irradiance
    gl_FragColor.a = 1.0;
  } else {
    gl_FragColor.a = 0.0;
  }
}
```

Listing 3: Convolution pixel shader

© The Eurographics Association 2008.

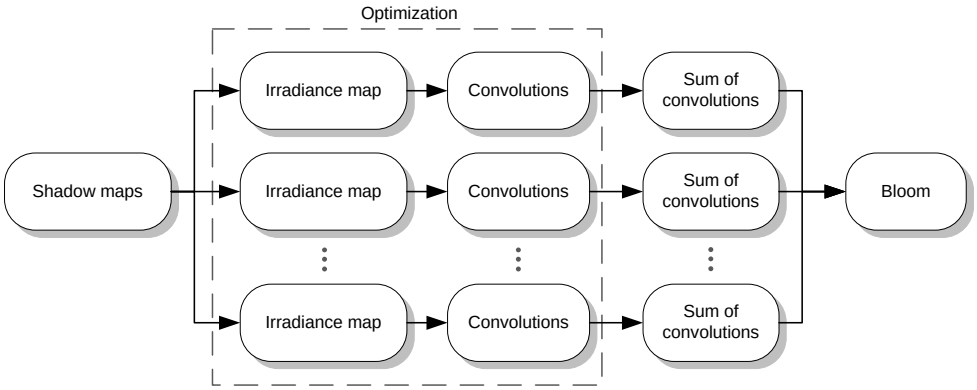

Figure 2: The five steps described in [DLE07]. We apply our three optimization techniques to the irradiance map and convolution processes.

Figure 3: From left to right: irradiance map for a given viewpoint as described in [DLE07]. Culled and clipped irradiance maps: occluded and clipped pixels are highlighted in blue and red respectively for visualization purposes. Final rendered image.

```
void main() {
  vec4 val = texture2D(irradiancemap,
                       gl_TexCoord[0].st);
  if (val.a > 0.0) {
    gl_FragColor = vec4(vec3(0.0), 1.0);
    // ... apply convolution kernel
  } else {
    gl_FragColor = vec4(vec3(0.0), 0.0);
    // ... no convolution
  }
}
```

In OpenGL, it would seem obvious to implement this optimization using the depth buffer as a mask, as described in [HB05], given that in modern graphics hardware pixels are discarded before the pixel shader is executed[†]. This depth buffer could be obtained during the computation of the irradiance map, and then transferred to the different convolution frame buffers. Irradiance maps of the same size could share the same depth buffer, thus limiting the number of necessary copies.

[†] For the specific card used in this work, the nVIDIA G-Force 8800, this technology is called *early-z* and *z-cull*.

However, we have observed that the alpha channel implementation still performs about 10% better than the shared depth buffer strategy. This may be due to the improvements in branching efficiency in the NVIDIA GeForce 8800 used [nVI06]. We thus opt to use the alpha channel instead, and perform an additional per-pixel check on it before convolving with the Gaussian functions.

4. Second Optimization: depth-based Irradiance Map

The main idea of our second proposed optimization is to adjust the size of the irradiance map according to the depth of the object being rendered. In the optimal case, rasterizing algorithms obtain each pixel value once per frame. However, several pixels in texture space (with a fixed texture resolution) may collapse into one final pixel in image space. This results in lots of wasted calculations. It is therefore convenient to modulate the size of the irradiance map according to the viewing distance of the object.

As opposed to mip-mapping techniques, where distances are considered individually for each pixel, we need to compute only one distance, which will be used for all the pixels in the current frame. We apply a conservative metric, adjusting the distance according to the pixel nearest to the camera.

© The Eurographics Association 2008.

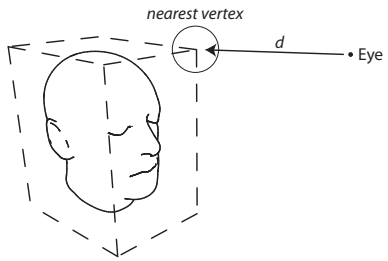

Figure 4: Distance from the camera (eye) to the nearest vertex in the bounding box.

If that pixel is visualized correctly, the rest (further away) will be as well. For fast computation of this distance, we approximate it by the distance d to the nearest vertex of the object's bounding box. This is shown in Figure 4.

Once d is found, we obtain the size T of the irradiance map on a per-frame basis by using:

$$s = min(\frac{d_{ref}}{d}, 1.0) \quad (3)$$

$$T = sT_{max} \quad (4)$$

where T_{max} is the maximum texture size, d_{ref} is the reference distance for which the irradiance map should have a size of $T_{max} * T_{max}$ pixels. In our implementation we have used values of $d_{ref} = 4.5$ (this value is dependent of both model scale and frame resolution) and $T_{max} = 2048$, as used by d'Eon and colleagues [DLE07].

Implementation issues: Allocating frame buffer memory is a costly process, so performing this operation on the fly adjusting its size according to the obtained T_{max} value would be impractical. Thus, we only allocate memory once for the maximum size T_{max}, the same way we would do without this optimization, and use the OpenGL function `glViewport(0,0,T,T)` instead while performing irradiance map calculations. During the convolutions and final rendering phases, instead of working with texture coordinates $(0..1)$ we limit texture space to the range $(0..s)$.

5. Third Optimization: Clipped Irradiance Map

In a traditional rendering pipeline, those parts of the model outside the view frustum would be logically clipped, and no computations would be performed on them. The method proposed in [DLE07], however, computes the complete irradiance maps for the complete model, regardless of which parts lie beyond the screen limits. This is justified because during the irradiance maps computations each vertex is mapped to texture space, and thus conventional clipping cannot be performed on them: their positions in texture space no longer determine their visibility in the final render step.

We avoid this problem by proposing an additional clipping operation independently of the standard view frustum clipping, to be performed during irradiance maps calculations. Given the following plane parameterization:

$$Ax + By + Cz + D = 0 \quad (5)$$

we obtain a user-defined view frustum by its six planes Π_i as follows (in clip coordinates):

$$\begin{aligned}
\Pi_1 &= (1,0,0,1) \\
\Pi_2 &= (-1,0,0,1) \\
\Pi_3 &= (0,1,0,1) \\
\Pi_4 &= (0,-1,0,1) \\
\Pi_5 &= (0,0,1,1) \\
\Pi_6 &= (0,0,-1,1)
\end{aligned}$$

As done in the first optimization, we could use a value slightly greater than one for D, so that we calculate some offscreen pixels that may have impact on the final pixels of the visible surface due to scattering. However, we have not observed any visible artifacts in our tests, and thus we use $D = 1$.

Figure 5 shows the traditional OpenGL pipeline: given that the coordinates of each vertex are already obtained in eye coordinates, transforming them to clip coordinates would mean a costly, per-vertex multiplication by the corresponding projection matrix. Instead, we transform the planes Π_i in Equation 6 to eye coordinates, and perform our clipping there. Let P be the matrix which transforms a vertex from eye to clip coordinates; we can transform the vectors defining Π_i by applying [SWND97]:

$$\Pi_{eye} = ((P^{-1})^{-1})^{\mathrm{T}}\Pi_{clip} = P^{\mathrm{T}}\Pi_{clip} \quad (6)$$

Figure 3 shows the resulting clipped pixels of the irradiance map.

Implementation issues:

OpenGL allows a user-defined clipping operation on top of the automatic clipping performed for the view frustum. The clipping planes Π_i are defined by using `glClipPlane`. However, this function transforms its parameters to eye coordinates by default, by using the modelview matrix. It is then necessary to assign the Identity matrix to the modelview matrix, before defining each clipping plane. We use the special output variable in the vertex shader called

© The Eurographics Association 2008.

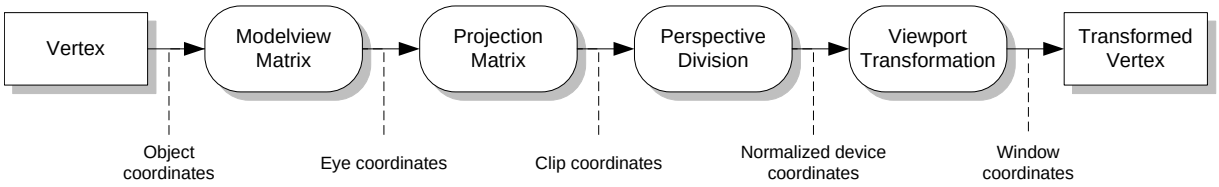

Figure 5: The OpenGL pipeline. We perform an additional, user-defined optimization in eye coordinates instead of clipping coordinates.

gl_clipVertex to specify clipping coordinates independently from the transformed vertex position. These will be used to perform the clipping operation with the clipping planes defined in `glClipPlane`.

Listing 4 shows the code for the modified part of the vertex shader for the irradiance map:

Listing 4: Irradiance vertex shader

```
void main() {
  vec4 pos = gl_ModelViewMatrix * gl_Vertex;
  // ... calculate irradiance with pos
  gl_ClipVertex = pos;
}
```

We initialize the alpha channel to 0 before computing the irradiance map. Thus, after obtaining it, all the clipped pixels will still have a zero value in the alpha channel, since they are not written by the pixel shader. This has the desirable effect of propagating the clipping to the Gaussian convolutions, given that, as explained in the first optimization, the alpha channel is checked before any convolution takes place.

6. Results

We have implemented the technique proposed by d'Eon and colleagues [DLE07], modifying it to include the three optimization techniques presented in this paper. We have used a GeForce 8800 GTX on a Core 2 Duo @ 2.4Ghz. Screen resolution was fixed at 1920x1200 pixels. The head model contains 25K triangles with 2048x2048 color and normal maps. We use six irradiance maps, with a maximum resolution of 2048x2048, the same as for shadow maps. Figures 6 and 7 show the results, rendered in real-time. Figure 8 shows a side-by-side qualitative comparison with the off-line technique by Donner and Jensen [DJ05] and the real-time shader of d'Eon et al. [DLE07]: visual inspection yields similar quality in the final renders. We could not duplicate the exact settings for our rendering given that some necessary data like light position or intensity is not included in the previously mentioned papers. Thus, some slight differences in tone are expected.

Table 1 shows the results of our tests, rendering images similar to Figure 9 with the three optimizations described in the paper. We varied the number of aligned heads from one to sixteen, as well as the distance (measured as the distance from the camera to the center of the middle head).

Figure 8: From left to right: images rendered by Donner and Jensen [DJ05], d'Eon et al. [DLE07] and our optimized algorithm, for comparison purposes.

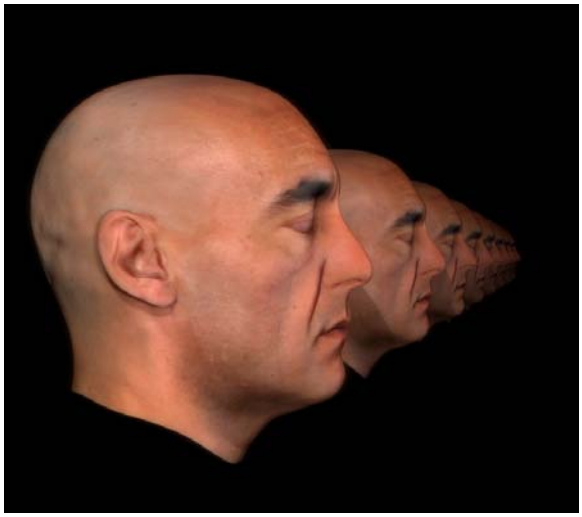

Figure 9: Example image used in our tests.

Our optimization techniques shows great scalability, achieving greater speed-ups as the number of heads increases. This makes them specially useful in multiple-character situations. Our depth-based irradiance map technique has a greater impact as the objects move away from the camera, with the opposite behavior for the clipped irradiance map optimization. All together, we achieve speed-up factors between 1.10 and 2.77. We have additionally performed similar tests using an older GeForce 8600M GS: in that case the speed-up factor with respect to the non-optimized version of the shader was even better, between 1.23 and 6.12.

© The Eurographics Association 2008.

Figure 6: Real-time rendering using two point light sources and two bloom passes.

Figure 7: Additional real-time renderings using two point light sources and two bloom passes.

Table 1: Speedup as distance and number of heads increases

	Distance					
Heads	**2.5**	**5**	**10**	**20**	**50**	**100**
1	1.17	1.10	1.39	1.48	1.53	1.61
2	1.40	1.10	1.46	1.63	1.88	1.88
4	1.66	1.39	1.58	1.89	2.10	2.26
8	1.86	1.67	1.95	2.01	2.34	2.59
16	2.01	1.90	2.28	2.41	2.50	2.77

7. Conclusions and Future Work

Our proposed optimization techniques are based on simple but effective ideas, and can be easily implemented on top of the work by d'Eon and co-workers. We achieve speed-up factors of up to 2.77 on a GeForce 8800 GTX, and up to 6.12 on the 8600M GS. The optimizations scale very well as the number of objects increases, making them suitable for crowd simulations, games or any other multiple-character scenario.

We have not made use of translucent shadow maps [SD03] since scaling the technique for more than one light source is potentially costly. As they are implemented in [DLE07], they take up to 30% of the rendering time, so the performance of our proposed techniques would be expected to drop similarly, since it does not optimize that part of the process. However, it is unclear whether the work described in [DLE07] uses them for both light sources to render their images. Their influence is limited to a number of specific situations, such as a thin translucent surface being lit from behind, although admittedly in those cases they can simulate very appealing subsurface scattering effects that we fail to capture (see Figure 10, left). However, for most cases, the results without translucent shadow maps are very convincing, even in extreme situations where subsurface scat-

© The Eurographics Association 2008.

Figure 10: Images rendered with our optimized algorithm without translucent shadow maps. Left: Scattering within the ear is not captured properly. Middle: Same image as it appears in [DJ06] for comparison purposes. Right: Backlit profile with dominant subsurface scattering still looks plausible.

tering dominates most of the simulated light field (see Figure 10, right). Even though their influence is marginal under most situations, efficiently implementing translucent shadow maps for multiple lights is an interesting direction of future work which we are currently working on.

8. Acknowledgments

The authors would like to thank XYZRGB Inc. for the high-quality head scan and Josean Checa for his all-around help. This research has been funded by the projects UZ2007-TEC06 (University of Zaragoza) and TIN2007-63025 (Spanish Ministry of Science and Technology). Jorge Jimenez was funded by a research grant from the Instituto Tecnológico de Aragón, while Diego Gutierrez was additionally supported by a mobility grant by the Gobierno de Aragon (Ref: MI019/2007).

References

[BL03] BORSHUKOV G., LEWIS J. P.: Realistic human face rendering for "The Matrix Reloaded". In *ACM SIGGRAPH 2003 Sketches & Applications* (2003), p. 1. 2

[DJ05] DONNER C., JENSEN H. W.: Light diffusion in multi-layered translucent materials. *ACM Trans. Graph 24*, 3 (2005), 1032–1039. 2, 3, 6

[DJ06] DONNER C., JENSEN H. W.: A spectral BSSRDF for shading human skin. In *Proc. of Eurographics Symposium on Rendering* (2006). 2, 8

[DL07] D'EON E., LUEBKE D.: Advanced techniques for realistic real-time skin rendering. In *GPU Gems 3*, Nguyen H., (Ed.). Addison Wesley, 2007, ch. 14. 2

[DLE07] D'EON E., LUEBKE D., ENDERTON E.: Efficient rendering of human skin. In *Proc. of Eurographics Symposium on Rendering* (2007). 2, 3, 4, 5, 6, 7

[HB05] HARRIS M., BUCK I.: GPU flow-control idioms. In *GPU Gems 2*, Pharr M., (Ed.). Addison-Wesley, 2005, pp. 547–555. 4

[HK99] HANRAHAN P., KRUEGER W.: Reflection from layered surfaces due to subsurface scattering. In *Proc. ACM SIGGRAPH '99* (1999), pp. 164–174. 1

[HV04] HAO X., VARSHNEY A.: Real-time rendering of translucent meshes. *ACM Trans. Graph 23*, 2 (2004), 120–142. 2

[INN05] IGARASHI T., NISHINO K., NAYAR S. K.: *The Appearance of Human Skin*. Tech. rep., Columbia University, 2005. 1

[JB02] JENSEN H. W., BUHLER J.: A rapid hierarchical rendering technique for translucent materials. *ACM Trans. Graph 21*, 3 (2002), 576–581. 1, 2

[JMLH01] JENSEN H. W., MARSCHNER S. R., LEVOY M., HANRAHAN P.: A practical model for subsurface light transport. In *Proc. ACM SIGGRAPH '01* (2001), pp. 511–518. 1, 2, 3

[MKB*05] MERTENS T., KAUTZ J., BEKAERT P., REETH F. V., SEIDEL H. P.: Efficient rendering of local subsurface scattering. *Computer Graphics Forum 24*, 1 (2005), 41–50. 2

[NRH*77] NICODEMUS F. E., RICHMOND J. C., HSIA J. J., GINSBERG I. W., LIMPERIS T.: Geometrical considerations and nomenclature for reflectance. National Bureau of Standards, 1977. 1

[nVI06] NVIDIA CORPORATION: nVIDIA GeForce 8800 GPU Architecture Overview, 2006. 4

[SD03] STAMMINGER M., DACHSBACHER C.: Translucent shadow maps. In *Proc. of the Eurographics Symposium on Rendering* (2003), pp. 197–201. 2, 7

[SKS02] SLOAN P. P., KAUTZ J., SNYDER J.: Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments. *ACM Trans. Graph 21* (2002), 527–536. 2

[Sta01] STAM J.: An illumination model for a skin layer bounded by rough surfaces. In *Proc. of the 12th Eurographics Workshop on Rendering* (2001), pp. 39–52. 1

[SWND97] SHREINER D., WOO M., NEIDER J., DAVIS T.: *OpenGL Programming Guide*, second ed. Addison-Wesley, 1997. Appendix F. 5

[TS67] TORRANCE K., SPARROW E.: Theory for offspecular reflection from roughened surfaces. *Journal of the Optical Society of America 57* (1967), 1104–1114. 2

[WTL05] WANG R., TRAN J., LUEBKE D.: Allfrequency interactive relighting of translucent objects with single and multiple scattering. *ACM Trans. Graph 24*, 3 (2005), 1202–1207. 2

[WWD*05] WANG L., WANG W., DORSEY J., YANG X., GUO B., SHUM H. Y.: Real-time rendering of plant leaves. *ACM Trans. Graph 24*, 3 (2005), 712–719. 2

© The Eurographics Association 2008.