Real-Time Skin Rendering on Graphics Hardware

Pedro V. Sander[*] ATI Research
David Gosselin[†] ATI Research
Jason L. Mitchell[‡] ATI Research

1. Introduction

We present a real-time algorithm for skin rendering which was used in the real-time animation *Ruby: The DoubleCross*, appearing in this year's SIGGRAPH animation festival. Our approach approximates the appearance of subsurface scattering by blurring the diffuse illumination in texture space using graphics hardware. This approach, based on the offline skin rendering technique proposed by Borshukov and Lewis, gives a realistic look and is both efficient and easy to implement. We describe algorithms to efficiently implement this technique in real-time using graphics hardware, as well as several enhancements to improve quality.

2. Texture-space skin rendering

The algorithm proceeds as follows:

1. Render diffuse illumination to a 2D light map
2. Compute and store shadows in the light map
3. Blur the light map
4. Render final mesh using the blurred 2D light map to provide diffuse illumination

Steps 1 and 4 are straightforward to implement using pixel shaders and renderable textures. We will describe how to compute shadows and how to perform the blur operation. Finally, we will describe additional acceleration techniques.

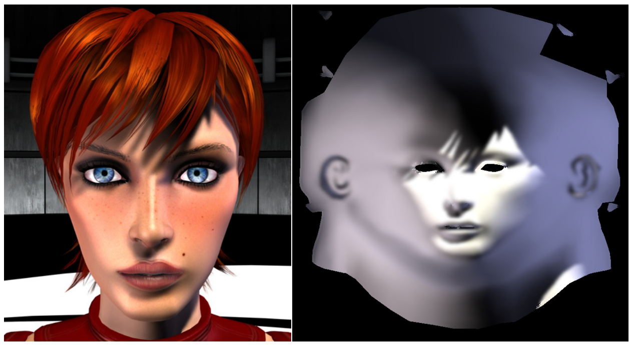
3. Soft shadows

Using this texture-space skin rendering technique, the computation of soft shadows is relatively inexpensive. A shadow map algorithm is used to determine visibility from the light, and texels in the diffuse light map are dimmed accordingly. The blur operation performed in step 3 will not only create the appearance of subsurface scattering, it will also provide soft shadows at no additional cost. The blur pass also significantly reduces aliasing, making it practical to use the shadow map algorithm.

Translucent shadows. Translucent shadows can be computed using a hybrid projective shadow algorithm. For example, to compute translucent shadows of the hair on the skin, we render the character's hair to the shadow map after the face has been rendered. We turn on the z test, but we turn off z writes. We write the colored opacity to the RGB buffer with additive blending (The z value for the opaque shadows can be stored in the alpha channel.) Then, when computing the shadows from the shadow map, if the sample is not shadowed by an opaque object, its diffuse contribution is attenuated by the value in the opacity values stored in the RGB channels.

Shadows on specular illumination. Since it is fairly expensive to perform an independent blur pass for the shadow component, we cannot independently apply shadows to the specular illumination. We have found, however, that the luminance of the blurred light map can be used to attenuate the specular term of the shadow casting light to obtain a natural look.

[*] E-mail: psander@ati.com
[†] E-mail: gosselin@ati.com
[‡] E-mail: jasonm@ati.com



4. The Poisson disc blur

We implemented the blur operation in hardware by using a pixel shader that applies a variable sized Poisson disc filter to the character's skin light maps in texture space. Using two temporary buffers, we performed several blurring passes on the diffuse illumination in order to achieve a soft, realistic look.

Variable kernel size. The kernel size used by our filter can vary in texture space. A scalar multiplier for the kernel size is stored in an 8-bit texture. Borshukov and Lewis addressed translucency on the model's ears by ray-tracing. We cannot afford to do ray-tracing in real-time, but we obtain a similar result by increasing the kernel size on the region around the ears.

Texture boundary dilation. In order to prevent boundary artifacts when fetching from the light map, the texture needs to be dilated prior to blurring. We needed an efficient real-time solution to this problem. We accomplished this by modifying the Poisson disc filter shader to check whether a given sample is just outside the boundary of useful data, and if so, copy from an interior neighboring sample instead. We only need to use this modified, more expensive filter in the first blurring pass.

5. Additional acceleration techniques

In order to optimize our technique, we employ hardware early-z culling to avoid processing certain regions of the light map.

Frustum culling. Before the blurring step, we clear the z buffer to 1 and perform a very simple and cheap texture space rendering pass in which we just set the z value to 0 for all rendered samples. If the bounding box of the model's head lies outside the view frustum, it is culled by our graphics engine, and thus the z value is not modified. On all further texture-space passes, we set the z value to 0 and the z test to "equal". This ensures that if the model lies outside the view frustum, hardware early-z culling will prevent all samples from being processed.

Backface culling. Backface culling can be performed by first computing the dot product of the view vector and normal. Then, if the sample is frontfacing, a 0 is written to the z buffer, thus culling the sample in all further passes.

Distance culling. If the model lies far from the camera, the z value is set to 1, and the light map from the previously rendered frame is used. Note that this does not affect specular lighting.

References

BORSHUKOV G. AND LEWIS, J. P. 2003. Realistic Human Face Rendering for "The Matrix Reloaded". *Sketches, SIGGRAPH 2003.*