REAL-TIME SKIN SHADER IN GLSL

Massimo Maj
www.mixmax3d.it

ABSTRACT

This paper presents an approach through OpenGL shading language for the rendering of human skin for game development. This method works in texture space so it requires an initial work for the creation of the images that are used for the simulation of the sub surface scattering effect.
The final image consists of a weighted average of the input images, the algorithm calculate this average trying to simulate the effect of the light scattering through the layers of the skin.

1. INTRODUCTION

The light model used for the calculation of the light contribution is based on the classic per-pixel bump mapping algorithm used in the example of the RenderMonkey software library, that simulate one point directional light. The vertex shader of the pipeline calculates per-pixel Light Direction and View Direction working on the position, normal, bi-normal and tangent of the geometry.

```
ViewDirection.x = dot( Tangent, ViewDirection );
ViewDirection.y = dot( Binormal, ViewDirection );
ViewDirection.z = dot( Normal, ViewDirection );

LightDirection.x = dot( Tangent, LightDirection.xyz );
LightDirection.y = dot( Binormal, LightDirection.xyz );
LightDirection.z = dot( Normal, LightDirection.xyz );
```

Those values are passed to the fragment shader.

2. INPUT TEXTURE

While I have been searching for previous works on this argument I found that a current approach used in the major 3d software like Autodesk Maya they use a three layered model for the calculation of the skin shader, the model is a combination of a dermal layer, a epidermal layer and a sub-dermal layer. The three layer can be drawn using a painting package using the diffuse/dermal layer as a base and adjusting the final color using a filter approach.

Figure 1 The Three layers of the skin with the normal map for specular calculation rendered on a HD geometry of a human head

Figure 2 The three images painted for the simulation: diffuse/dermal, epidermal and sub-dermal.

3. GAMMA CORRECTION

First of all each color contribution is linearized through gamma correction for an appropriate use on different devices. A dedicated function in the fragment shader makes the work on each texture sampler.

```
Final_color = pow(Color,2.2);
```

4. THE OPENGL LIGHTING MODEL

According to the OpenGL specifications the lighting model works on three terms to simulating the light : ambient, diffuse and specular. In this method each term is a color in the RGB format and is multiplied for each layer of the skin. A physical method for the calculation of skin sub surface scattering must works

according to the laws of conservation of energy and to the laws of Fresnel for reflection and refraction of light. The ambient, diffuse and specular terms in the equation of light are calculated according to these laws taking into account each layer of the skin.

```
ambient_color = (Cdiffuse*0.5* Fresnel * fvAmbient
            + Cscatter1*0.4* Fresnel * fvAmbient
            + Cscatter2 *0.1* Fresnel * fvAmbient );
```

```
diffuse_color = ((
Cdiffuse*0.5 * Fresnel * fvDiffuse * fvLightColor
+ Cscatter1*0.4* Fresnel * fvDiffuse *fvLightColor
+ Cscatter2 *0.1* Fresnel * fvDiffuse *fvLightColor) )
* lerp(vec4(1.0), Cdiffuse*lum, 0.1);
```

```
specular_color = (Cdiffuse*0.5 * fvSpecular
+ Cscatter1*0.4* Fresnel * fvSpecular
+ Cscatter2 *0.1* Fresnel * fvSpecular ) +
((specularity));
```

Each term is based on a weighted average of the input textures.

```
Cdiffuse * 0.5 + Cscatter1*0.4 + Cscatter2 *0.1;
```

For the specular term in the sub surface scattering simulation a Gaussian Noise texture is used as a noise factor on the bump calculus to add realism.

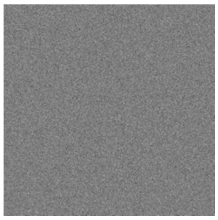

Figure 3 The Gaussian Noise Texture

For the specular term in the sub surface scattering simulation a Gaussian Noise texture is used as a noise factor on the bump calculus to add realism.
The pure specularity of the skin must be white, and it is calculated separately and multiplied by the noise texture.

```
specularity =      fvSpecular * ( pow( fRDotV,
fSpecularPower ) ) * fNDotL * Cnoise ;
```

The final color is a sum of ambient color, diffuse color, specular color and the specularity.

```
final_color = ambient_color + diffuse_color +
specular_color + specularity ;
```

Figure 4 The Real time Rendering of this algorithm : Front view.

Figure 5 The Real time Rendering of this algorithm : Rear view.

Figure 6 The Real time Rendering of the basic concept of this algorithm inside a WebGL implementation.

5. CONCLUSIONS

This paper introduce an algorithm for real time rendering of the human skin in texture space for game development. Future works can explore the possibility for the calculus of translucent surfaces such as the ears and nostrils of the nose, the implementation of a skin shader inside an HDR pipeline simulation and the exploration of a screen space method for the sub surface scattering calculation .

6. REFERENCES

1. Fabien Houlmann, Stéphane Metz, High Dynamic Range Rendering in OpenGL;

2. Larry Gritz, Eugene d'Eon, GPU Gems 3, Chapter 24. The Importance of Being Linear;

3. Eugene d'Eon, David Luebke, GPU Gems 3 Chapter 14. Advanced Techniques for Realistic Real-Time Skin Rendering;

4. Borshukov, G; Lewis, J. P. (2005). "Realistic human face rendering for "The Matrix Reloaded"". Computer Graphics (ACM Press);

5. Eugene d'Eon, E (2007). "Advanced Skin Rendering". GDC 2007;

6. Jimenez, J., Sundstedt, V., and Gutierrez, D. 2009. Screen-Space perceptual rendering of human skin. ACM Trans. Appl. Percept. 6, 4, Article 23 (September 2009), 15 pages.