Real-time Rendering of Parametric Skin Model

Stanislav Fecko*
Supervised by: Martin Madaras†

Faculty of Mathematics, Physics and Informatics
Comenius University in Bratislava
Bratislava / Slovakia

Abstract

When a ray of light enters a translucent material, it starts to move through an optically non-uniform environment, resulting in scattering of the light under the surface of the object. After traversing the material, the light eventually leaves the object. This happens not at the point where it entered though, but in slightly different. Visual effect called the sub-surface scattering is often visible on materials like marble or wax. Human skin is formed of multiple layers of different skin tissues. That is why effects, usually seen on other translucent materials also occur on the skin. If we want to render highly realistic skins, we need to include this knowledge into our lighting model. For this we have developed a real-time method using various techniques used in existing papers. We use an offscreen buffer, into which we render the entire object in texture-space and perform the diffusion to simulate the effect of sub-surface scattering on human skin. Our aim is to extend this method even further. We are going to make a model capable of rendering also non-human skins, often used on sci-fi or fantasy characters. Their skins have qualities that human skin does not and thus we need special approach to their realistic rendering.

Keywords: Real-time rendering, Skin rendering, Sub-surface scattering, Texture-space Diffusion, Deferred Shading, Screen-space Ambient Occlusion

1 Introduction

We present here our approach to highly realistic real-time skin rendering. We describe the methods used in our pipeline as well as both their advantages and disadvantages. Our rendering system uses the concept of the deferred shading, supports high dynamic range rendering and takes advantage of the methods like texture-space light diffusion, screen-space ambient occlusion or environment mapping technique.

Our aim here is to design and implement a working rendering system, that, apart from rendering human skin, is capable of rendering also various non-human characters. These can be often seen in games or movies and to render them appropriately, we need to use special methods. We present techniques we have used and show the results these allowed us to achieve.

2 Related work

We are using the concept of the deferred shading in our program. It is widely used in various real-time applications to avoid a loss of performance on shading hidden pixels. The approach also allows the using of many local lights and directly supports numerous image-based post-processing effects. The idea of the deferred shading is rather old already, first time presented (although not yet using the 'deferred shading' name) by Michael Deering [3]. Implementation in the Killzone 2 computer game, as well as many interesting details are described in [13]. The concept and its limitations are explained in a presentation by Hargreaves and Harris [12]. Koonce in his chapter in GPU Gems [9] describes different extensions and effects available when using deferred shading.

An important part of the project is the simulation of the sub-surface scattering effect. There are different approaches to rendering translucent objects. An offline method described in an article by Donner and Jensen [5] is based on the photon mapping technique, which takes about 15 minutes (on an Intel Core 2 Duo 2.4GHz) to generate the resulting image, but it is capable of producing such advanced effects as volumetric caustics, translucent inter-scattering between surfaces and volumetric shadows. A rapid hierarchical integration of irradiance computed at selected points on the surface is used in [7]. Depending on the model, it can deliver results in under one minute. The technique presented by D'Eon et al. [4] uses texture-space diffusion to compute the scattered lighting in real time. Shah, Konttinen, and Pattanaik [11] use dual light-camera space technique, computing the area integral via a splatting approach in image-space. Also GPU Gems have a chapter on rendering of this phenomenon [6]. The fast methods do not produce result of the same quality as the previously mentioned techniques, but their huge advantage is the speed making the methods suitable for real-time rendering applications.

There are also various methods used for estimation of

*stanislav.fecko@gmail.com
†martin.madaras@gmail.com

ambient occlusion. For static scenes this factor can be precomputed for fast later use. Every decent modeling software is capable of baking the ambient occlusion maps and the process usually takes seconds to minutes. For dynamic objects, however, these maps can not be used and the occlusion has to be computed every frame. In offline rendering this factor can be obtained by ray-tracing, which gives good results, but the more complex the scene is, the longer it takes to estimate the occlusion. Mittring describes in his article [10] a real-time method they used when developing the computer game Crysis. This calculation is done in screen-space, needs no preprocessing and is independent from the geometric complexity of the rendered scene. Other screen-space implementations are described in an article by Bavoil and Sainz [1], where the authors face the precision issues of existing real-time ambient occlusion methods. Bavoil, Sainz and Dimitrov [2] introduce another similar technique for the estimation of the ambient occlusion based on image-space information.

3 Sub-surface scattering

Sub-surface scattering (SSS) is a natural phenomenon observed on the translucent materials. Light entering the volume of the object interacts with the material, scatters, and exits the volume at a different point and a different direction. This causes even the areas that are not directly lit, to receive some amount of lighting. The effect is most notable in the scenes where an translucent object is lit by a back light. Despite the fact that a perfectly opaque object would be black from our point of view, the translucent object will have a bright silhouette due to the light that traversed through the material and left on the other side. The effect is very well visible on materials like wax or milk.

A human skin is a much more complex material than it may look. It consists of many thin layers with different optical properties. This is why the light penetrating the skin is scattered under the surface, resulting in the very same effect described earlier.

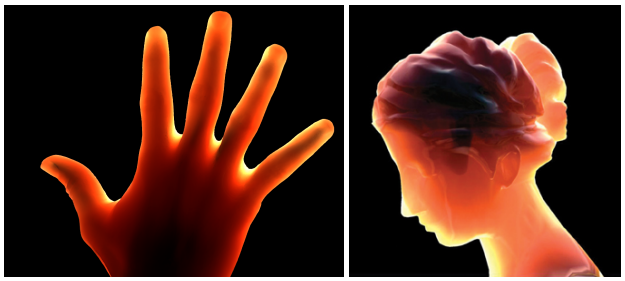

Figure 1: Sub-surface scattering on translucent materials. The first image is from [5], the second is from [6].

3.1 Sub-surface scattering via texture-space diffusion

The method proposed in [4] approximates the effect of the sub-surface scattering using techniques affordable in real-time rendering. The idea is to render the entire scene into a off-screen buffer, process this data and then, in the second pass, use it to render the final image.

First phase of the algorithm is rendering of the object into the texture space. To perform this task, a special vertex shader is used. It projects each vertex (x,y,z) from the world space to its texture coordinates (u,v), therefore into the texture space. The result of this pass is a shaded texture, or rather a lightmap, of the entire object. It contains light information for every point on the surface of the model, including the areas that are not in a viewing volume of the camera, nor the light volume of the light source. Important note is that only the ambient and the diffuse components of the Phong's lighting model are used in this light calculation.

The second phase is where the the data is processed, as mentioned earlier. This is actually the part where the simulation of the sub-surface diffusion takes place. We blur the light intensity values captured on the surface of the object, making the bright spots to bleed into the darker areas. Therefore, also the points with little direct light can be lit, provided that they are close to some bright ones. In reality, the metric for this 'closeness' is a three-dimensional distance of two points. Our approach uses a two-dimensional distance in the texture space. This difference leads to different results, but the fact that we can compute this value very quickly outscores the precision drawback.

The second phase consists of a few blurring shader passes applied on the lightmap. What is important here is the fact that all this blurring is done in the texture space. Therefore the result is also, as in the first stage, a lightmap. After the blurring, these are blended together forming one resulting lightmap carrying all the information about the ambient and the diffuse light.

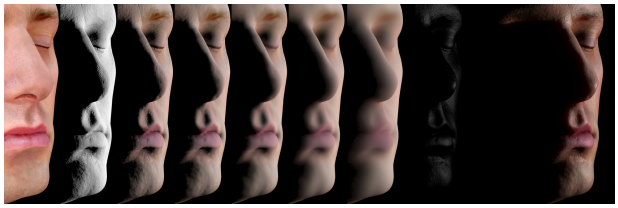

Figure 2: Image from [4] describing the phases of the sub-surface scattering rendering process. From left to right: Albedo; Diffuse light component; Blurred versions with different kernel size used; specular highlights and the final composition.

The third phase requires another geometric pass. Here we render the whole scene again, but instead of computing every pixel's brightness, we use the blurred lightmap. The diffuse and the ambient components of the pixel's il-

lumination are read from the lightmap, and the specular component is added for the pixel separately.

We compute specular highlights separately due to the fact that the specular component of the Phong's illumination model represents the light reflected from the surface. That is why this light does not interact with the deeper material structure and is not scattered, merely mirrored off the topmost layer. The fact that we treat this component apart from the other two results in a soft translucent look of the rendered objects while the fine details on their surface are visually preserved thanks to the specular highlights.

4 Our approach

Our implementation of the skin-rendering pipeline is done in C++ and OpenGL. Because of the features we use to achieve our results we require OpenGL 3+ capable hardware. We are making use of the framebuffer objects, which allow us to easily take advantage of the deferred shading concept in all stages that require the lighting, HDR rendering and high-presicion image manipulation, as well as the screen space ambient occlusion effect.

Default OpenGL pipeline outputs its results into the backbuffer, which is a memory segment used for collecting the pixel values of the final image. However, OpenGL 3 gives the programmer the opportunity to change this default behavior of the pipeline and exchange the backbuffer with a special target called framebuffer object. This feature allows us to redirect the output of our rendering into textures. The same textures that we can later read from as from any other textures. This functionality has a wide range of applications, amongst others also the HDR rendering, the deferred shading or the texture-space diffusion effect.

4.1 Our extension for non-human skins

Our aim in this paper has been to extend the concept also for the non-human skins. The texture-space diffusion technique gives the nice results for human skin, because it gives it the look of a translucent material. However, there are situations when we need to render also the non-human characters like aliens, undead, dragons and others, with highly realistic features. This is where we may need to extend the method to fit the particular type of skin.

The model used for the testing of our rendering techniques is the head of Davy Jones, a character from a well known movie. The mesh and the texture were created and are the property of Luis Manuel Morillo [1]. This character model suits our needs because of its rather specific skin, which is pretty much like the skin of an octopus - therefore wet, slimy and oily. Since human skin rarely has such attributes, we had to slightly extend the method for human skin rendering to achieve the satisfactory results with our Davy Jones.

[1] http://luima.com

We added the environment mapping feature, which is a rather quick method used to approximate the reflections on the object. It does not give the actual mirrored image, rather relies on the fact that the user is not focused on particular details of the reflected light. It uses a texture, which it maps onto the surface of the scene objects. We will describe the feature in detail in section 4.2.3.

4.2 Implementation details

4.2.1 Texture-space diffusion

In the first phase we are going to map triangles from the world space into the texture space. Therefore our output framebuffer ought to be resized to the dimensions of the texture. Due to the performance load of working with resolutions over 4096x4096 pixels, we decided to use the framebuffer sized up to 2500x2500 pixels. This reduces the quality only slightly while significantly improving the speed of the rendering.

Figure 4: Bumpiness parameter of the skin. The left column has the value 0, middle column 0.75 and the right one is rendered with the parameter set to 3. Top row is without color and environment mapping, the bottom row are complete renders.

Since we are using the deferred shading approach, we do not render the intensity values into the buffer straight away. First we transform each vertex to its texture coordinates and into the buffer write only its world-space position and normal. Note that in this pass we do not need to capture per-pixel depth information. Since we are using the bump-mapping effect on the surface of our model, we are performing this normal vector transformation in this phase. Our calculation uses a greyscale texture as a heightmap in the tangent-space, from which the world-space normal is determined every frame. This process is controlled by a bumpiness parameter describing how much the heightmap affects the final normal vector. The lowest valid parameter value of 0 corresponds to unchanged normal, therefore a smooth surface. The higher the value is, the more featured the bumps become. In all figures in this paper, except for the Figure 4, the value 1 was used.

Only after the entire scene is rendered, we proceed to the shading. For every light source one pass is required

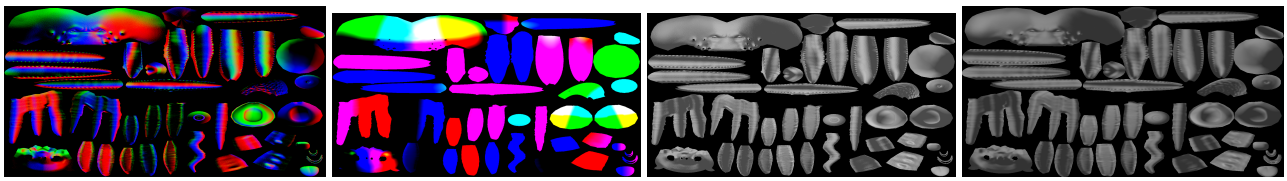

Figure 3: Content of the framebuffer textures. From left to right: world-space normal, world-space position, computed lighting (ambient and diffuse) and the last is the blurred version of the previous.

and the additive blending is used for merging the separate passes. The result of this pass is a texture containing the light intensity values for every texel of the original texture.

Figure 5: Amount of blurring and it's impact on the look of the surface. From left to right used kernel size: 3x3tx, 8x8tx and 13x13tx. Top row contains only ambient and diffuse light components, the bottom one is also with specular highlights and SSAO.

The second phase uses the separable gaussian blur to simulate the diffusion process. The amount of blurring depends on the skin parameters. The more translucent look we desire, the wider kernel has to be used. Currently the skin is parameterized by one real number describing the radius of the blurring effect. After the two shader passes we have the resulting blurry lightmap in one of the framebuffer's textures (see Figure 3). This texture can be, at the beginning of the phase three, bound to the texture unit and, the same way as any other texture, used as a source. Finally, in the last part of the algorithm we use the light intensity information contained in the processed texture to determine the ambient and the diffuse components of the pixel's illumination. After adding the specular highlights in a separate pass, the diffusion method is at an end (see Figure 14).

4.2.2 Screen-space ambient occlusion

Screen-space ambient occlusion (SSAO) is a technique used in real-time rendering to achieve more realistic looking scenes by darkening the areas where only limited amount of light can get due to the geometric obstacles in the point's neighborhood. The proper ambient occlusion factor is viewer-independent, therefore it can be precomputed and stored with the model in a separate texture. This approach allows for the very high quality results, but if the scene changes, the occlusion factors change as well, and therefore the precomputed data cannot be used.

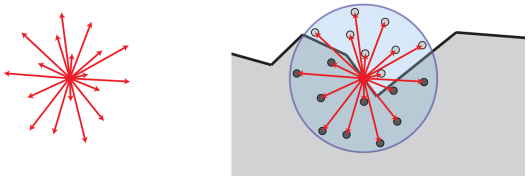

Figure 6: Left: static set of sample positions. Right: Positions applied on the examined pixel's surroundings.

A screen-space approach trades accuracy for speed. The results produced by this technique are no longer viewer-independent, tend to be noisy and suffer from various aliasing-related artifacts. The advantages of the method are the speed and the versatility. Occlusion can now be produced in real-time, for any dynamic scene, regardless its geometric complexity. With a few optimizations, the results are produced quickly and with rather sufficient quality.

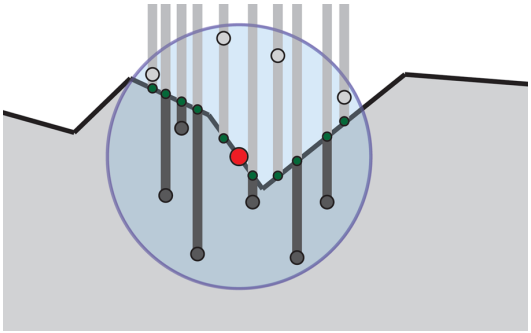

Figure 7: Sampling pixel's neighborhood and comparing the depths of the samples against the scene geometry. The dark circles contribute to the examined pixel's occlusion, while the bright circles decrease the amount of the occlusion. The green circles correspond to the values in the depth map.

The main idea of the method is sampling every screen pixel's neighborhood and by comparing the depth values of these samples we determine the amount of the occlusion the sampled points cause onto the examined pixel. Every sample that falls behind the geometry of the scene

(its depth value is greater than the value in the depth map) contributes to the examined pixel's occlusion (pixel will be darker) and every sample closer than the depth map reduces the occlusion (brightness increases).

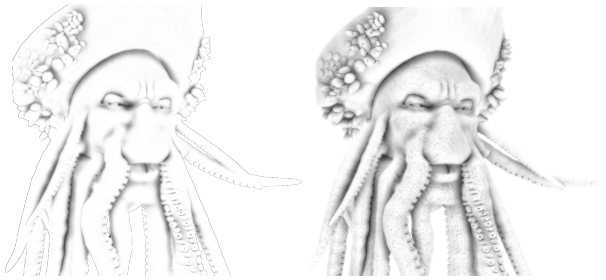

Figure 8: Screen-space ambient occlusion using only depth information (left) compared to the method using per-pixel screen-space normals (right). In both cases the number of taken samples was the same. Note the difference caused by the bump-map.

A technique taking advantage of the screen-space normals is often used to both speed up the ambient occlusion computation and make it more accurate. Provided that we have a normal vector in the examined pixel, we can choose to sample only the points inside the hemisphere in the normal's direction. This way we need merely half of the samples we would have to use otherwise. Moreover, due to the fact that the normal vectors are projected into the screen-space after the normal/bump mapping takes place, the method generates the ambient occlusion also on flat surfaces enhanced by bump-mapping. Since the bump-mapping is computationally pretty cheap effect, this improvement of the ambient occlusion estimation algorithm is particularly useful for applications requiring a high framerate. Also scenes containing rather low-polygon models are efficiently rendered with satisfactorily good results.

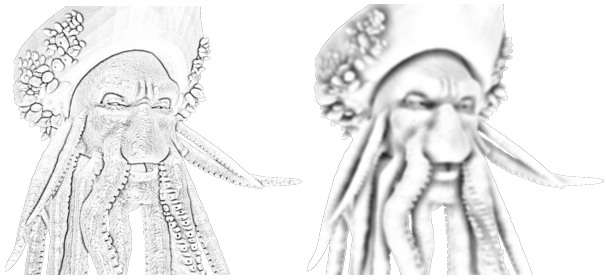

Figure 9: High (left) and low-frequency (right) ambient occlusion factors.

$$\mathrm{SSAO} = \sqrt[a+b]{(\mathrm{High})^a \cdot (\mathrm{Low})^b} \qquad (1)$$

We use a static set of sample positions, but to avoid artifacts, we rotate this set by different angle for each pixel. This way we trade artifacts and visible patterns over the image for a rather uniform noise. Afterwards we reduce the noise by blurring the occlusion map a little. To achieve the occlusion on both small details and larger areas, we use one sampling pattern for high-frequency features and a different one for low frequency. When combined together, these give a satisfactory quality of the occlusion, while the cost of the effect is affordable.

In our implementation of ambient occlusion we used different parameters for high and low-frequency maps. Figure 9 shows the intermediate results. In the first case, 16 samples were taken and then the texture was blurred by 3x3 pixels gaussian blur. In the low-frequency case, the number of samples was 28 and the size of the blur was 15x15 pixels. The mixing formula we use for blending these two maps is in Equation 1. We achieved best results with parameters $a = 1$ and $b = 2,42$.

Figure 10: Final screen-space ambient occlusion map determining the amount of the ambient light at the pixel.

4.2.3 Environment mapping

To achieve wet or slimy look of the rendered surfaces, we added the effect called environment mapping. It approximates mirror reflections on the objects in the scene. The method requires a environment texture, which is usually a panoramatic photograph and describes the surroundings of the object or the entire scene. The technique uses every scene point's normal vector, reflects the viewer-to-pixel vector from the surface and computes the reflected direction vector. This direction determines which texel from the environment texture is to be used. This value is then added to the pixel. The method is in a combination with the deferred shading very fast, requires only one full-screen shader pass that reads only one filtered texel value for every screen pixel.

This mechanism is described in the Figure 11. The environment texture, which encircles the scene objects, is

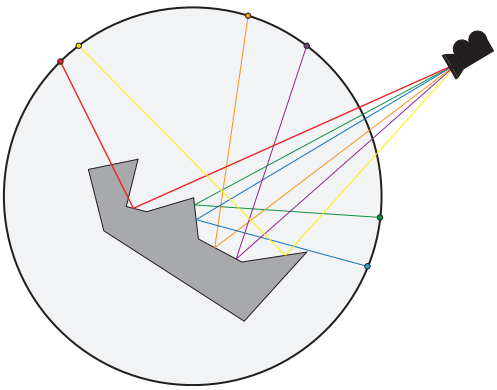

Figure 11: Concept of environment mapping.

drawn here as a thick black circle. Ray reflects from the object's surface and from the point where it hits the background we sample the texture. This color is then multiplied by a reflectance of the skin and added to the current pixel's value. We use the reflectance parameter to control how much environment illuminance is reflected by the skin, therefore to determine how much (on scale 0 to 1) the surrounding scene affects the object's look. In Figure 12, this value is 0.75, while in Figure 13 the used value of the parameter is 0.1.

Note the thick red ray, which after the reflection crosses the volume of the object on its way to the background. Properly calculated ray would reflect a few more times before reaching the environment texture. However these bounces are ignored in order to receive a fast rendering method.

Since this approach does not take self-reflections into consideration, it may produce unconvincing results when used on complex objects. And as our testing model could be considered somewhat complex due to the numerous tentacles, we had to deal with this problem. Our easy and fast solution uses the result of the SSAO procedure and modifies the amount of the reflection based on the amount of the occlusion. The idea is that the more occluded the pixel is, the more reflections the ray has to undergo to reach it. Therefore, its energy is strongly reduced and that is why we decided to reduce its contribution.

5 Performance and results

In Equation 2 is described how the effects are combined into the final image composition. First, the line 1 takes place in the framebuffer (in texture-space; all the other lines are in the screen-space). During the second geometry pass, lines 2 and 3 are evaluated (in that order). Last two lines describe the application of the environment mapping and the final fusion of all the effects. Meaning of the High and Low, as well as a and b is shown in the Figure 9. Parameter Skin describes the properties of the particular type of skin. Ambient, Diffuse and Specular are the components of the widely used Phong's shading model. Values EnvMap and Albedo are color vectors read from textures.

Figure 12: Environment mapping enhanced by the SSAO term. Left is untreated, right is with the improvement. The reflection intensity is intentionally very high to make the difference more visible.

$$\begin{aligned} \text{AmbDif} &= (\text{Ambient}\cdot\text{Skin}_{\text{amb}} + \text{Diffuse}\cdot\text{Skin}_{\text{dif}})_{\text{blur}} \\ \text{SSAO} &= \sqrt[a+b]{\text{High}^a\cdot\text{Low}^b} \\ \text{SSS} &= \text{AmbDif}\cdot\text{SSAO} + \text{Specular}\cdot\text{Skin}_{\text{spec}} \\ \text{Reflection} &= \text{EnvMap}\cdot\text{Skin}_{\text{refl}}\cdot\text{SSAO} \\ \text{Final} &= \text{SSS}\cdot\text{Albedo} + \text{Reflection} \end{aligned} \tag{2}$$

Figure 13: The final image output of our rendering system.

Our skin rendering pipeline has been tested on Windows 7 Home Premium system with GeForce GTX560 graphics card, Intel Core i7-950 processor and 12GB of physical memory. When all effects are switched on and the application runs in 1920x1080 resolution with no multisampling, we achieve average pace of the rendering above 20fps. As we use many screen-space and texture-space techniques, the time required to process one frame strongly depends on the screen and the texture resolutions. But since this is still a work in progress, we expect to introduce certain optimizations and to improve the frame rate.

Table 1 presents the performance of the application based on two parameters: the geometric complexity of the

Figure 14: Sub-surface scattering process on our model. From left to right with additional effects: ambient and diffuse lighting; blurred lighting; with screen-space ambient occlusion; with specular highlights.

Figure 15: Composition of the effects. From left to right: Sub-surface scattering alone; SSS with environment mapping; SSS with screen-space ambient occlusion; SSS with both environment mapping and ambient occlusion.

Mesh complexity	Screen resolution	Framerate
65 K	640 x 480	63
65 K	800 x 600	63
65 K	1280 x 720	47
65 K	1920 x 1080	31
259 K	640 x 480	46
259 K	800 x 600	35
259 K	1280 x 720	31
259 K	1920 x 1080	21

Table 1: Performance of our solution.

scene and the screen resolution. Since the pipeline consists of two geometric passes, we have to draw over half a million polygons per frame (with the more complex mesh), what also reflects on a lower framerate in scenes with the higher complexity of the geometry.

As is also clear from both the table and the description of the used methods, time needed to render one frame strongly depends on the used resolution. This is why cases with lower screen dimensions scored double the framerate of the scenes with higher resolution. However, in all used scenarios we still reach real-time rates, therefore satisfying results. Even in the most complex scenes and the highest resolutions we rarely dropped below 20fps. Here the average was slightly over 21 frames per second.

6 Conclusion

The result of our project is a C++ class dedicated to real-time skin rendering. It bases on the deferred shading concept utilizing the benefit of the HDR rendering. It implements the sub-surface scattering effect via texture-space diffusion technique, estimates the ambient occlusion using a fast, screen-space approach and simulates the surface reflections by the environment mapping method. At its current state it delivers real-time results even for the complex model we are using.

There are also effects we have not implemented yet and features we would like to add to the project as a future work. Our plan is to create a procedural skin generator for an automatic production of various non-human skins. We are also searching for and experimenting with the implementation of other real-time methods that would improve the translucent and slimy look of the rendered character while retaining the high framerate.

The area we are now mainly focusing on is building a texture generator capable of producing different types of skin textures as well as bump or reflection maps. Our idea is to use various pre-made images and by blending these to create new ones. The prototype uses the framebuffer objects and OpenGL shaders to process the source images and output the final textures based on a given description. We aim to generate these skins in real-time. This way the application could be used for both designing and rendering of the characters.

7 Acknowledgements

We would like to express our thanks to Luis Manuel Morillo for providing us both visually nice and geometrically complex model of Davy Jones's head that we have been using.

References

[1] Louis Bavoil and Miguel Sainz. Multi-layer dual-resolution screen-space ambient occlusion. In *SIGGRAPH Talks*. ACM, 2009.

[2] Louis Bavoil, Miguel Sainz, and Rouslan Dimitrov. Image-space horizon-based ambient occlusion. In *ACM SIGGRAPH 2008 talks*, SIGGRAPH '08, pages 22:1–22:1, New York, NY, USA, 2008. ACM.

[3] Michael Deering, Stephanie Winner, Bic Schediwy, Chris Duffy, and Neil Hunt. The triangle processor and normal vector shader: a vlsi system for high performance graphics. In Richard J. Beach, editor, *SIGGRAPH*, pages 21–30. ACM, 1988.

[4] Eugene d'Eon, David P. Luebke, and Eric Enderton. Efficient rendering of human skin. In Kautz and Pattanaik [8], pages 147–157.

[5] Craig Donner and Henrik Wann Jensen. Rendering translucent materials using photon diffusion. In Kautz and Pattanaik [8], pages 243–251.

[6] Simon Green. Real-time approximations to subsurface scattering. In Randima Fernando, editor, *GPU Gems: Programming Techniques, Tips and Tricks for Real-Time Graphics*, chapter 16. Pearson Higher Education, 2004.

[7] Henrik Wann Jensen and Juan Buhler. A rapid hierarchical rendering technique for translucent materials. *ACM Trans. Graph.*, 21(3):576–581, 2002.

[8] Jan Kautz and Sumanta N. Pattanaik, editors. *Proceedings of the Eurographics Symposium on Rendering Techniques, Grenoble, France, 2007*. Eurographics Association, 2007.

[9] Rusty Koonce. Deferred shading in tabula rasa. In Hubert Nguyen, editor, *GPU Gems 3*, chapter 19, pages 429–457. Addison-Wesley, 2008.

[10] Martin Mittring. Finding next gen: Cryengine 2. In *SIGGRAPH '07: ACM SIGGRAPH 2007 courses*, pages 97–121, New York, NY, USA, 2007. ACM.

[11] Musawir A. Shah, Jaakko Konttinen, and Sumanta N. Pattanaik. Image-space subsurface scattering for interactive rendering of deformable translucent objects. *IEEE Computer Graphics and Applications*, 29(1):66–78, 2009.

[12] Mark Harris Shawn Hargreaves. Deferred shading. https://developer.nvidia.com/-sites/default/files/akamai/gamedev/docs/-6800_Leagues_Deferred_Shading.pdf.

[13] Michal Valient. Deferred rendering in killzone 2. Online, accessed Feb. 20th, 2012, 2007. Develop Conference, http://www.guerrilla-games.com/publications/dr_kz2_rsx_dev07.pdf.