Principles of Rendering: Example short paper

Richard Southern*
National Centre for Computer Animation

Abstract

This document serves to demonstrate the short paper format used for the Principles of Rendering unit. This submission should be no longer than 2 pages, inclusive of references and images, but excluding appendices and code listings. You should aim for between 5 and 8 references in your bibliography. An abstract provides an executive summary of your short paper.

1 Introduction

In this section you should introduce the shader effect you're trying to recreate. You should include a few examples of real reference images which you are trying to simulate, such as in Figure 1. You must also decompose the chosen effect into individual layers or effects which you are going to try to implement.

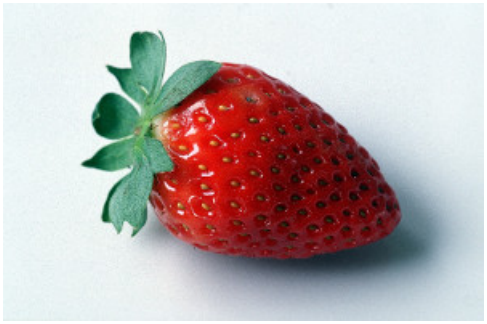

Figure 1: *An example reference image, acquired from [free-pictures-photos.com 2016].*

For example, the strawberry in Figure 1 might exhibit:

1. Regular seed placement which causes small indentations in the surface.
2. A gentle colour gradiation from tip to top.
3. Narrow specular highlights.
4. Shallow subsurface scattering properties.
5. Soft shadowing.
6. Slight depth of field effect.

You might also use this section to introduce examples of how this effect may have been created in the past, particularly in books, industry talks or academic papers, where the method used to recreate the effect is known. Make sure you make use of references here, such as [Akenine-Möller et al. 2008] or [Pharr and Humphreys 2010] in order to contextualise your work.

2 Method Overview

In this section you should present the final method which you have implemented in a manner that is as independent of the language used to create the effect as possible. There are a number of different ways to do this, for example using a mathematical formulation as in Equation 1.

$$L_o(\mathbf{x}, \omega_o, \lambda, t) = L_e(\mathbf{x}, \omega_o, \lambda, t) + \int_\Omega f_r(\mathbf{x}, \omega_i, \omega_o, \lambda, t) L_i(\mathbf{x}, \omega_i, \lambda, t)(\omega_i \cdot \mathbf{n}) d\omega_i \quad (1)$$

Alternatively you might want to present your method in the form of a pseudocode algorithm, as in Algorithm 1. You will probably also find that data flow diagrams[1] are very useful in conveying how your shader is implemented using OpenGL when multiple render passes are needed.

Algorithm 1 Euclid's algorithm

```
1: procedure EUCLID(a, b)                    ▷ The g.c.d. of a and b
2:     r ← a mod b
3:     while r ≠ 0 do                        ▷ We have the answer if r is 0
4:         a ← b
5:         b ← r
6:         r ← a mod b
7:     return b                              ▷ The gcd is b
```

You should avoid describing your method using code examples if possible. If you must, an example of code inclusion and markup is provided in Appendices A and B.

3 Results

Here you should present the rendering results of the method(s) which you have described in Section 2 above. Ideally, you should provide a breakdown of the different layers or render passes used in the construction of the final effect (see Figure 2 for an example). Where the geometry is complex, it might also be suitable to render a simple object (a sphere perhaps) to demonstrate the effect of the different render layers. If you are demonstrating a dynamic effect, you might want to render several frames of a sequence.

(a) *Diff+Spec* **(b)** *Diff+Spec+Ambient*

Figure 2: *A side by side comparison of different render effects is always a good idea to demonstrate a choice of parameters or design of your shader. These were appropriated from [Renderman Support 2016].*

If you are generating a real time effect, it might also be suitable to demonstrate with some numbers or statistics the performance improvements that you were able to attain by optimisations to your shader or C++ application.

*e-mail:rsouthern@bournemouth.ac.uk

[1] See `https://en.wikipedia.org/wiki/Data_flow_diagram`.

You may also want to provide a discussion of development iterations and the thought processes by which you arrived at the final method presented in Section 2. It is important in this section to be self-reflective and critical: we don't expect you to have created the perfect effect. If you are disatisfied by any aspect of the result, you should communicate this to us with some sort of explanation.

References

AKENINE-MÖLLER, T., HAINES, E., AND HOFFMAN, N. 2008. *Real-Time Rendering (3rd edition)*. A.K. Peters Ltd.

FREE-PICTURES-PHOTOS.COM, 2016. Fruits large free pictures for print. https://www.free-pictures-photos.com/fruits/.

PHARR, M., AND HUMPHREYS, G. 2010. *Physically Based Rendering: From Theory to Implementation*. Elsevier.

RENDERMAN SUPPORT, 2016. Writing and compiling a simple shader. https://renderman.pixar.com/view/simple-shader-writing.

A Renderman Shader Example

An example of Renderman code listing is included below. Please make sure that when including code segments that you include only sections that are directly relevant to the application at hand, and preferably which link to the method described in Section 2.

Listing 1: *Renderman example lifted from [Renderman Support 2016].*

```
surface basicSpecular(
    color myOpacity = 1;
    float roughness = 0.1;
)
{
        color myColor = (1.0, 0.0, 0.0);
        normal Nn = normalize(N);
        //Specular stuff
        vector V = normalize(-I);

        Ci = myColor * myOpacity * diffuse(Nn) +↩
             specular(Nn, V, roughness);
        Oi = myOpacity;
}
```

B GLSL Shader Example

An example of a GLSL code listing is included below. Note that C++ code listing is also supported using the same method.

Listing 2: *A simple textured shader.*

```
// The texture coordinates
smooth in vec2 o_TexCoord;

// This is passed on from the vertex shader
in vec3 LightIntensity;

// The texture to be mapped
uniform sampler2D u_Texture;

// This is no longer a built-in variable
out vec4 o_FragColor;

void main() {
    // Set the output color of our current pixel
    o_FragColor = vec4(LightIntensity,1.0) * ↩
        texture(u_Texture, o_TexCoord);
}
```