Creating and Rendering a Chocolate Ball using RenderMan

Andy Abgottspon[*]
National Centre for Computer Animation, Bournemouth University, 2011

Abstract

For the RenderMan part of the CGI Tools assignment, a simple real world object had to be re-created and rendered. The tasks were to analyse the object and apply the correct techniques in terms of displacement, shading, lighting, etc. This report documents these steps and the methods used. In order to produce two final images, a Ferrero Rocher chocolate ball has been created from scratch and placed in a scene.

1 Introduction

Photographs have been taken for reference and in order to analyse the object. These pictures have been shot in various locations to get a better understanding of the impact of lighting and the environment (see Figure 1).

Figure 2 illustrates the stages in the development of the shaders as described in the following chapters.

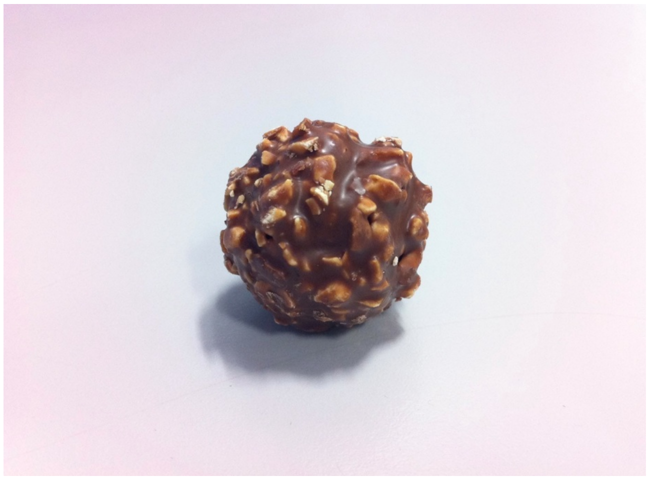

Figure 1: *Reference image of Rocher chocolate ball*

2 Primitive modelling

The original chocolate ball has a spherical shape but contains lots of bumps created by the little hazelnut pieces. In terms of modelling, a simple sphere has been used within RenderMan.

[*]e-mail: info@aaweb.ch

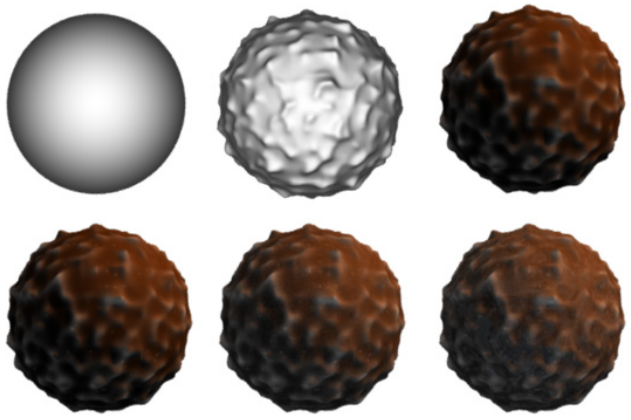

Figure 2: *Stages of development starting with a sphere, adding displacement and colour variation (top row). The bottom row shows the addition of little noise spots, grain and an environment map (left to right).*

The actual distinct shape was achieved using displacement, as further discussed in Section 4.

3 BRDF model

Chocolate – and this object in particular – has a matte surface, with low reflectivity but specular highlights. This means, the environment is barely reflected, while still affecting the overall colour slightly. The final colour is calculated as follows:

```
Ci = Oi * Ct * (
              magnitude * bumpColor +
              mixCol*noise + grainCol +
              Kd*diffColor*diffuse(Nn) +
              (Ks*spec + Kr*Cr)
             );
```

The diffuse term of the expression above is the classical Lambert reflection model. The other main components are *bumpColor* to change the colour based on a point's displacement (as described in Section 4), *mixCol* that is used for little noise spots and *grainCol* (see Section 5). Specularity and reflectance are further explained in Section 6.

4 Displacement

This was the most difficult part of the assignment. Looking at the reference material, it was obvious that the actual distribution of the hazelnut pieces as well as the resulting colour on the surface can vary immensely (see Figure 3).

To achieve this result, a number of steps were necessary. First of all, the displacement shader uses the noise function with a certain frequency to produce the humps. Furthermore, using the *pow* function, the shape of the noise pattern can be influenced.

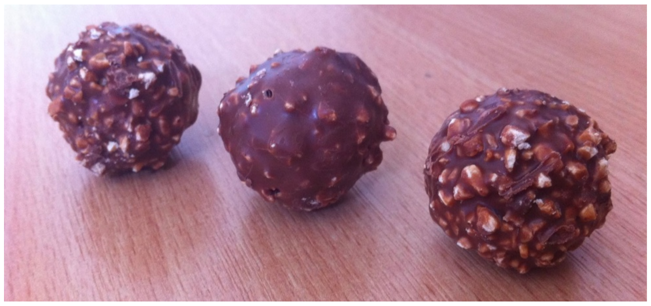

Figure 3: *Reference image showing variety in the different individual chocolate balls.*

The second part trick happens in the surface shader, where points that are more displaced are shaded differently. This is used to account for the very thin chocolate layer covering the little bits of hazelnut where the colour of the chocolate seems to be a lot brighter.

Technically speaking, this might have been achieved by transferring the magnitude from the displacement shader to the surface shader using an output variable (e.g. *output varying float dispMagnitude* followed by *displacement(”dispMagnitude”, myMag)* to retrieve the value in the surface shader). Another, maybe slightly hacky, option would have been to do all the displacement directly in the surface shader.

The approach chosen was to pass the values controlling the displacement to the surface shader and re-do the calculations from the displacement shader. This way, although computationally not optimal, different values could be chosen allowing for even more control and additional variation.

5 Natural variation and noise

The objects used for reference contained very little bright spots, caused by tiny air bubbles within the chocolate structure. These do not occur that frequently in the surface itself and were therefore used only sparsely as shown in the first picture of the bottom row of Figure 2.

To achieve this, a random noise value was generated for every point, but then only applied to the surface if it is greater than a certain threshold, allowing precise control over the amount of these little spots.

In a similar way, subtle grain was added to the overall surface as shown in step 5 (Figure 2, bottom row, centre). This helps getting rid of the slightly plasticky look. The grain uses a similar method as used for the little spots, but is applied to all points rather than certain areas.

The *generalSurface* shader also includes the effect mentioned above to allow any surface to have grain. This was used for the background images of the final renders, i.e. the table and wall.

6 Environment map

The map used was shot in my room and created from six images using the following command (*tdlmake* is 3delight command. For prman, use the command *txmake*):

```
tdlmake -envcube -fov 93 photo1.JPG photo2.JPG
    photo3.JPG photo4.JPG down.JPG up.JPG map.tx
```

Although the *ward anisotropic* model is typically used for metals, it provided pleasing results in this case (see stage 6 of Figure 2) since the reference object does not have a lot of reflectivity. The algorithm implemented is based on Jon Macey’s slides[1].

7 Scene

To produce two final images, a table and wall object have been added to the scene. The *generalSurface* also allows them to have a bit of grain which gives a more organic look to the surface. The camera was rotated to create a more dynamic angle.

For scene 1 (see Figure 4), only one light source is producing shadows. Scene 2 (see Figure 5) includes another light source recreating the conditions in my room where the environment maps and wood textures were captured. Particular attention has been paid to the shot composition, by applying the rule of thirds and using additional objects in the background to underline the depth of field effect.

The scenes were rendered using the free RenderMan compliant renderer *3delight*[2].

8 Depth of field

For this static object, motion blur was not very appropriate. Therefore, depth of field was chosen to mimic a camera artefact. This can be achieved very easily within the RIB file using the command:

```
DepthOfField 1.8 0.100 5
```

In this case, 1.8 is the focal length (how blurry things are) and 5 is the distance from the camera to the focus point, the focal distance.

9 Conclusion

Considering the complexity and variety of such an object, the overall result looks convincing. However, more tweaking and the control of additional parameters could be included to be able to achieve a very specific outcome (matching different styles like in Figure 3) rather than a more general one.

Reflecting on the assignment, finding the right functions and parameters to feed them can be quite tedious and time-consuming. On the other hand, it was a very pleasing experience to be in total control of all the aspects of an objects and images final look and understanding the techniques behind it.

Bibliography

Stephenson, I., 2007. *Essential RenderMan Fast – 2nd Edition*, London: Springer.

Cortes, R. and Raghavachary, S., 2008. *The RenderMan Shading Language Guide*, Boston: Thomson Course Technology PTR.

[1]http://nccastaff.bournemouth.ac.uk/jmacey/Renderman/index.html
[2]http://www.3delight.com/en/index.php?page=3DSP_overview

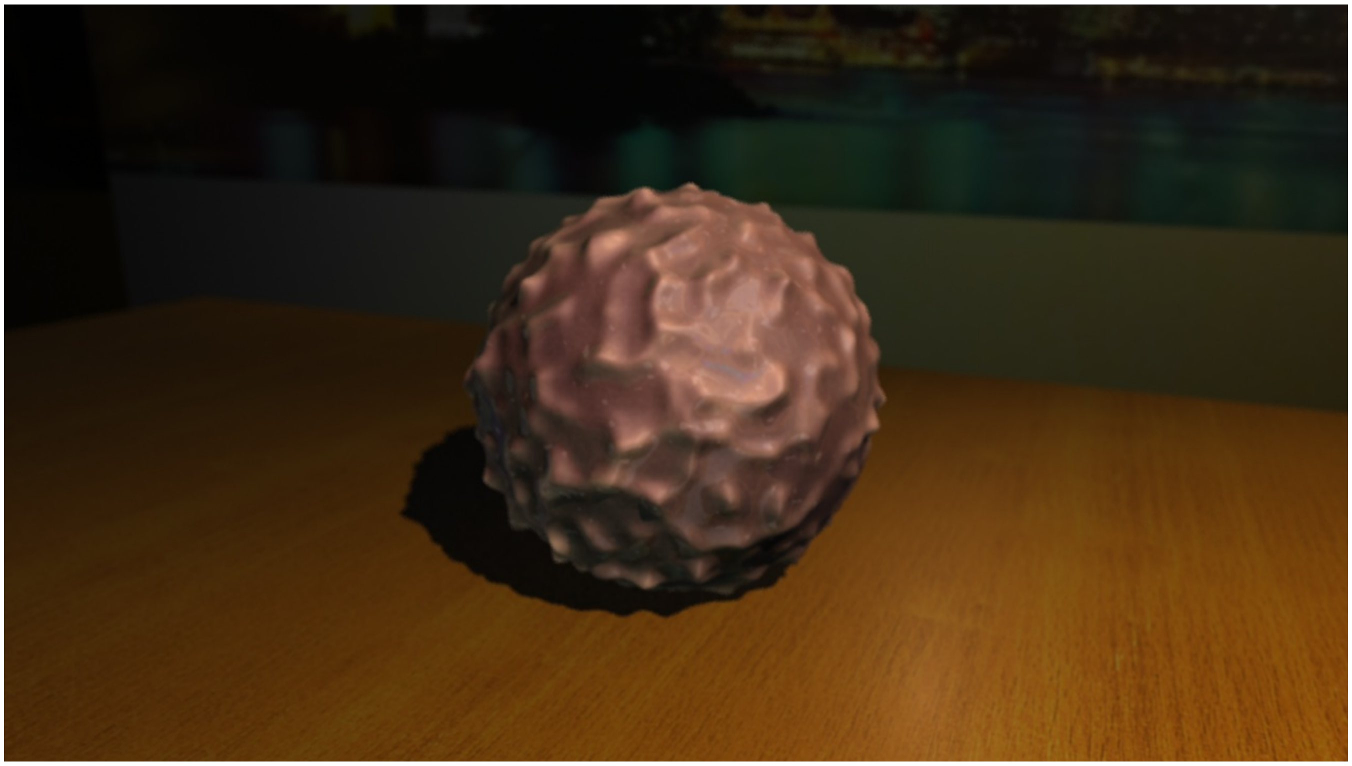

Figure 4: *Final render of Scene 1: Ferrero chocolate ball. Rendered using 3delight.*

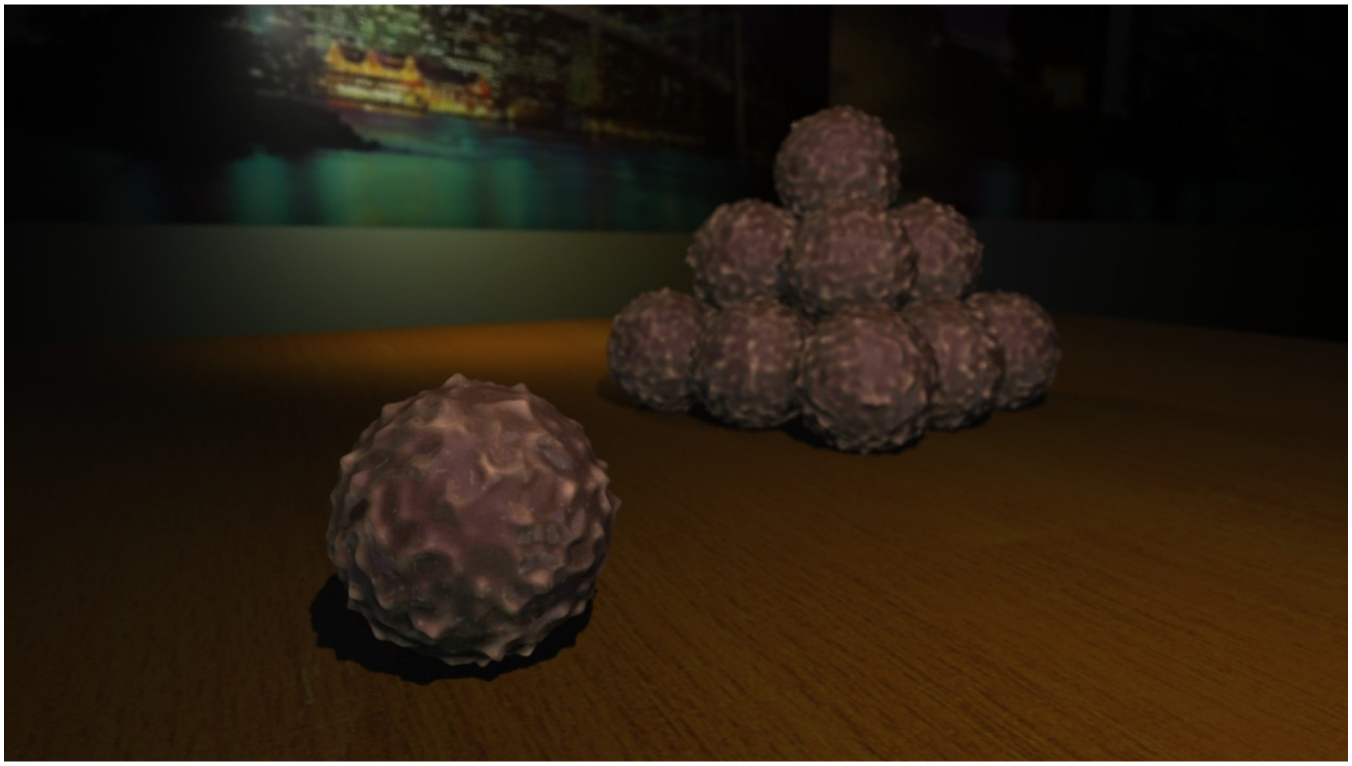

Figure 5: *Final render of Scene 2: Ferrero chocolate balls using two light sources and depth of field. Rendered using 3delight.*