Advanced Topics in Computer Graphics II

Universität Bonn
Institut für Informatik II
November 20, 2019

Winter term 2019
Prof. Dr. Reinhard Klein
Alexander Dieckmann, Lukas Bode

Sheet R3 - Material Comparison

Learning Based Material Estimation

For milestone 1 you have acquired a Ward SVBRDF of some material using the TAC7. Now, we want to compare this measurement to a SVBRDF predicted by the neural network of Deschaintre et al. (`https://team.inria.fr/graphdeco/projects/deep-materials/`) from a smartphone image with activated flash. Thus, please take a picture of the material you have used for milestone 1 with your phone. Since the position of the flash is usually very close to the camera itself, we have a semi-calibrated illumination situation, which can be leveraged by the network. So please make sure that the flash produces clearly visible highlights on the object in order to achieve good results. Note, that the smartphone should be parallel to the plane that the material sample resides in. The picture has to be scaled and cropped before being fed to the network. Since the network expects the picture to be taken with about 45 degree field of view, cropping alone is not sufficient. You can look at the networks README.txt file for more details.

Shading

The networks output consists of four png images: diffuse color, specular color, roughness, and normals. These images have to be loaded into your program using e.g. stb_image.h (code: `https://raw.githubusercontent.com/nothings/stb/master/stb_image.h`, tutorial: `https://learnopengl.com/Getting-started/Textures` in Loading and creating textures). After loading them, the shading can be done using the following algorithm:

```
Require: Incident lighting direction vectors ($\omega_L$), View direction vector ($\omega_V$), Material $M$ = [Diffuse map ($d$),
  Normal map ($n$), Specular map ($s$), Roughness map ($r$)]
  Performed at each surface point separately
  $h \leftarrow \text{Normalize}((\omega_L + \omega_V)/2)$                                ▷ Half-vector computation
  $d \leftarrow \frac{d(1-s)}{\pi}$                                                        ▷ Diffuse contribution scaling
  $D \leftarrow \frac{1}{\pi}\left[\frac{r^2}{(n\cdot h)^2(r^4-1)+1}\right]^2$              ▷ Cook-Torrance micro-facet normal distribution
  $G \leftarrow \frac{1}{(n\cdot\omega_L)(1-\frac{r^2}{2})+\frac{r^2}{2}} \frac{1}{(n\cdot\omega_V)(1-\frac{r^2}{2})+\frac{r^2}{2}}$   ▷ Cook-Torrance shadowing and masking term
  $F \leftarrow s + (1-s)2^{((-5.55473(\omega_V\cdot h))-6.98316)(\omega_V\cdot h)}$           ▷ Fresnel effect approximation
  return $\frac{(\frac{FGD}{4}+d)(n\cdot\omega_L)}{\omega_L.Z}$   ▷ Final rendering equation compensated for low angles lighting directions regarding
  $\omega_L$ and the upright normal direction
```

Figure 1: Pseudo code of shading. Taken from `https://repo-sam.inria.fr/fungraph/deep-materials/SupplementalMaterial_Single-Image_SVBRDF_Browser/PseudoCode.pdf`.

Fair Comparison Using OpenGL

A fair comparison of the measurements requires to see both of them side by side under the same illumination and viewing conditions. Thus, we want to have a single scene containing lighting, camera, and object(s) and want to shade it two times: Once with the TAC7 material applied to an object and once with the other material applied to the same object.

Rendering to Texture

In order to be able to store and display both shadings, instead of rendering to the screen directly, we now have to render to textures using a framebuffer. You can basically follow the tutorial at `https://learnopengl.com/Advanced-OpenGL/Framebuffers`. It is well written and will guide you through the process. The tutorial renders the scene in a texture and subsequently renders the texture to the screen. We need to render the scene two times to different textures. There are two ways, to approach this problem:

One way is to attach a single texture to the framebuffer, bind the framebuffer, bind the right shader, and draw the scene. This will store the result in the texture for later usage. Afterwards, you can attach another texture to the framebuffer, bind the other shader, and draw the scene again, resulting in the second rendered scene being stored in a texture.

The other way is to directly attach two different textures to the framebuffer and configuring the shader to write the result of the TAC7 material shading into the first attached texture and the result of the Deschaintre material shading into the second attached texture. The general rendering procedure of this method is the same as for the first one: Bind framebuffer, bind shader, and draw the scene. Using this technique, however, avoids to draw the scene twice.

You are free to choose which solution you want to implement.

Rendering Textures Side-by-Side

After writing the two shaded scenes to textures, the easiest way to display them on the screen is to draw two different quads, each filling one half of the screen. Each quad should have texture coordinates ranging from 0 to 1 in both dimensions. Thus, you can simply add another shader, that just makes a texture lookup and writes the result to the screen.

General Remarks

- Do not forget to attach a depth buffer to the framebuffer. Otherwise, it will not be able to perform a depth test.
- Before drawing the scene, you may have to adjust the viewport and the camera to fit the shapes of the quads you use to draw the texture to the screen.
- Do not forget to unbind any framebuffers if you want to draw to the screen. Otherwise, you will draw to the bound framebuffer instead.

Good luck !