INTERACTIVE GEOMETRY LAB

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Computer Graphics Laboratory ETH Zurich

Computer Graphics
Exercise 4 - BRDF and Procedural Textures

Handout date: 08.11.2013

Submission deadline: 29.11.2013, 14:00h

You must work on this exercise **individually**. You may submit your solution by:

- Emailing a .zip file of your solution to `introcg@inf.ethz.ch` with the subject and the filename as `cg-ex4-firstname_familyname`

The .zip file should contain:

- A folder named `source` containing your source code.
- A `README` file clearly describing what you have implemented for each module.

Your submission must be self-contained: include all GLSL shaders and HTML/CSS/JavaScript code required to run your solution. You will not need other third party libraries, but if so please justify its use in your `README` and include it in your submission as well. Do NOT include any binary files.

No points will be awarded unless, your source code can:

- Run on Firefox web browser on student lab PCs at CAB H 56 or 57 running Windows or Linux.

Your submission will be graded according to the quality of the images produced by your renderer, and the conformance of your renderer to the expected behavior described below for each module.

This exercise will be graded **during the exercise session on 29.11.2013**. You are required to attend, and those absent will receive 0 points. Grading will be conducted on the lab machines in room **CAB H 56** or **57**. You are also required to email your solution beforehand as described above.

Goal

The goal of this exercise is to implement various bidirectional reflectance distribution functions (BRDF) and procedural textures using OpenGL shading language (GLSL). A list of BRDF models and procedural textures is presented to you.

While we encourage you to implement all the modules in a common framework, the solutions for each module should be independent. Implement a separate shader program for each module, and structure your code so that each feature is independent from others. Your final solution will be able to render the given

scene using different BFDFs and textures and to change its behavior during its run-time using mouse and keyboard interaction.

Before getting started, it is highly recommended to read through the OpenGL Shader Language specification. You may find useful documentation about GLSL including the specification and the quick reference sheets here.

Grading

Your solution will be graded according to the quality of the result images and the conformance of your code to the requirements described here in the exercise sheet. In this exercise you must implement **all** modules.

If you have partial solutions in your submission, describe them clearly in your `README`, otherwise there will be no partial credit. Describe what you have implemented or tried to implement for each module.

As a last note, try to optimize your shaders so that they do not hurt the real-time interactivity.

Code Template

A simple WebGL code template is given to you, as well as template shader files. For each module you will need to create and implement a new shader program (vertex shader and fragment shader).

The code template provides you with an interactive web front-end, where you type in shaders and load them dynamically to see them working, and export for your submission. You can switch between different shaders by selecting one form the drop-down box. There are two material property sets you can switch between by pressing space bar. Use them for the two materials you will model in the BRDF part. You may export the current rendering by pressing 'e' key.

It is likely that your browser is not allowed to access local resources for security reasons. In order to avoid this problem, you may need to upload the entire code to your personal web storage (such as the one provided by ETH) and work remotely.

Camera Navigation

Trackball mouse navigation is provided in the sample code. Do not modify the mouse interaction for your final submission as we will need to navigate your scene for grading.

Bidirectional Reflectance Distribution Function

Implement each BRDF for the same scene presented in the code. Simple vertex and fragment shaders for ambient lighting are provided. Add shaders for other BRDFs described below. Your program should allow the user to change the BRDF dynamically through the drop-down box in the web interface.

Throughout the following modules, you need to model the look of the two materials, hematite and polished steel (Fig. 1(a) and (b)), using each BRDF. Some BRDFs may not be enough to model them. Discuss what are the reasons in your `README`. Your solution needs to be able to switch between two materials when the space bar is pressed. For the SVBRDF, model only one material described in the module.

(a) Hematite (b) Polished steel (c) Rusted steel

Figure 1: Surface materials.

Try various values for each model's parameters and find out the effects of those parameters. You should not hard code the parameter values in the shader, but pass them via uniforms or appropriate OpenGL functions. Maybe it will be useful to have a user interface to tweak the parameters, but this should not appear in the final image.

Most BRDFs consist of the ambient, diffuse, and specular components. The ambient and diffuse components do not depend on the viewing direction and hence their distributions are equal in all directions. They are well understood and treated in a similar way across various BRDFs. On the other hand, the specular component is view-dependent and are modeled in quite different ways. The following BRDFs are distinguished mostly by the way they model the specularity (if they do). Unlike the ambient component, the diffuse and specular components are associated with specific light sources. Do not forget to add up these components for all light sources.

2: Lambertian Model (5 points)

Lambertian model only represents perfect diffuse surfaces. Write shader programs implementing the BRDF for Lambertian model.

3: Phong Lighting Model (5 points)

Phong model can represent specularity as well as the diffuseness. Implement Phong lighting model using vertex and fragment shaders.

4: Blinn-Phong Model (5 points)

Blinn-Phong model is a slight modification to the Phong lighting model, and is the default reflection model used in OpenGL's fixed-function pipeline. Most notable difference is that Blinn-Phong model uses the halfway vector between the viewing (eye) direction and the direction to the light source. Then it uses the angle between the halfway vector and the normal instead of the angle between the viewing direction and the reflected direction of the light source. It is considered that Blinn-Phong model better represents empirical BRDFs than Phong model does. Implement Blinn-Phong model.

5: Ward's Model (10 points)

Ward's BRDF is an empirical model and designed to fit the reflectance measurement. It has been widely used in computer graphics as it has only a few parameters and is easy to control. Ward's BRDF as well as the Cook-Torrance BRDF below is based on the assumption of a surface as a collection of microfacets. Ward BRDF is defined as the sum of a diffuse term and a Gaussian anisotropic gloss lobe to model specularity. The specular lobe is defined by two parameters, which describe its widths in two principal directions of anisotropy. You may include as many specular lobes as you want.

Read the accompanied Ward's paper[1], and implement **anisotropic** Ward BRDF. Section 4.2 will provide you with relevant information. You will need to define a local tangent frame at each vertex given the normal at the vertex and the tangent direction you should provide. Define the tangent direction consistent across the surface. In his original publication, Ward presented a cheaper approximation of the specular term. Do not use this approximation, but try to find a cheaper, but exact computation.

6: Cook-Torrance Model (10 points)

Cook-Torrance BRDF is an example of the physics-based reflectance models. In this model, the specularity is explained by assuming that a surface consists of macrofacets, each of which reflects specularly. Only the portion of facets which have the normals in the halfway direction between the viewing direction and the direction of the light source contribute the specularity. Thus the specularity is modeled as a multiplication of; the Fresnel term F—the spectral reflectance property of the ideal specular reflection by a smooth microfacet; the facet slope distribution function D—the distribution of the microfacet's orientation in terms of the halfway direction; and the geometrical attenuation factor G—the geometric property between microfacets, such as shadowing and masking between them.

Read the accompanied Cook and Torrance's paper[2], and implement Cook-Torrance model. Pages 7–12 of the paper contain most information you need. Use as many specular lobes as you want. Do not consider the polarization.

Hints: The exact computation of the Fresnel term F is computationally expensive and dependent on many physical properties. Several approximations have been proposed for different materials, including the one presented by Cook and Torrance in their original publication. You may use any of them to compute Fresnel term F. One of the often used approximations is proposed by Schlick[3], where the reflectance is interpolated between the reflectance values at normal incidence and grazing incidence such that:

$$F(\lambda, \theta) = F_0(\lambda) + (1 - F_0(\lambda))(1 - \cos\theta)^5,$$

where λ is the wavelength, θ is the incident angle, and $F_0(\lambda)$ is the spectral distribution of the reflectance at normal incidence. The Fresnel term at glazing incidence ($F_{\pi/2}(\lambda)$) equals to 1 for all wavelength λ. The reflectance at normal incident can be either captured or computed using Fresnel's equation as:

$$F_0 = \left(\frac{n_1 - n_2}{n_1 + n_2}\right)^2,$$

where n_1 and n_2 are the refractive indices of the incident and refractive media. Note that the refractive index n also varies according to the wavelength λ. You may approximate the spectral distribution with RGB color components, so that you only need to define these functions for three components. You may use different reflectance at normal incidence F_0 or different refractive index n as a parameter for each

[1] G. Ward, "Measuring and modeling anisotropic reflection," in *Proceedings of SIGGRAPH '92*, 1992
[2] R. L. Cook and K. E. Torrance, "A reflectance model for computer graphics," *Transactions on Graphics*, vol. 1(1), 1982
[3] C. Schlick, "An inexpensive BRDF model for physically-based rendering," *Computer Graphics Forum*, vol. 13(3), 1994

color component. Assume the object is in vacuum. For the facet slope distribution function D, use the Beckmann distribution.

7: Spatially Varying BRDF (SVBRDF; 5 points)

For more realism, BRDFs can be defined with additional two parameters for the spatial coordinates, so that they can model spatial variations of the reflectance across the surface. However, because of its higher dimensionality, it is difficult to acquire the measurement or fit a model to it. To circumvent this, we take the idea of procedural texture. 2D texture coordinates can be acquired through the built-in attributes in the shaders. Modify **Cook-Torrance BRDF** so that its parameters—as well as the base color—vary across the surface. Try to make your steel surface look rusted as in Fig. 1(c). Make use of your Perlin Noise function (see below) to give the illusion of randomness.

Note

You may find various resources about the foregoing BRDFs. For clarification, it is usually a good idea to refer to the original publications, which in most cases can be found online through the publishers or the conference organizations, or by web searching. It is generally encouraged to do some 'research' by trying to find literature and understand it. You should be able to access digital libraries offered by most publishers and organizations within ETH networks.

Procedural noise

Change the surface of the object using the following textures. Enhance your **Phong** shader to implement the following modules. Again, your program should allow the user to interactively switch between modules.

A: Wood grain (10 points)

Implement a 3D Perlin Noise function. Use this function in your shader to produce a 3D wood texture that is visually similar to Fig. 2(a).

(a) Wood (b) Marble

Figure 2: Image of real wood and real marble.

B: Marble (10 points)

Use your Perlin Noise function to produce a 3D marble texture that is visually similar to Fig. 2(b).

C: Earth (20 points)

Use Perlin Noise to implement an "Earth like" appearance on your model. The appearance should have the following characteristics.

A large portion of the surface should be oceans: a near constant, smooth blue surface[4].

The other portion should be scattered islands of various sizes. These land masses should have deserts and grassy plains which affect the color. Also use your Perlin Noise function to *bump map* your islands with scattered mountains[5].

Finally there should be a scattered assortment of white clouds that float on the surface slowly **over time**. Adapt your Perlin Noise function to allow you to change the color on your surface based on the elapsed time. Thus your renderer will no longer show a static scene but an animation.

Notes

Ken Perlin has a nice assortment of Java Applets that use Perlin Noise on his website, specifically look under "textures" and "worlds". He has also posted an online presentation that explains Perlin Noise in 3D and 4D, and gives some hints hows to manipulate the raw noise function into useful textures.

[4] Consider what effect an absolute value or clamping function would have when applied to your noise function
[5] What would happen if your 3D noise function also affected the normals in your shader?