Computer Graphics
Exercise 6 - Procedural Textures

Handout date: 28.11.2014

Submission deadline: 12.12.2014, 14:00

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 both the *subject* and the *filename* as `cg-ex6-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 12.12.2014**. You are required to attend, and those absent will receive 0 points. Grading will be conducted on the lab machines at **CAB H 56**. The students whose last name starts with L–Z should come to the lab by 14:15, whereas those whose last name starts with A–K should do by 15:00. You are also required to email your solution before the submission deadline written above. Late submissions will not be considered, and will be given 0 points even if they have been graded during the grading session.

Goal

The goal of this exercise is to implement various Procedural Textures using Perlin noise. You will use OpenGL shading language (GLSL), like you did for Exercise 4 about BRDF.

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 textures and to change its behavior during its run-time using mouse 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.

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

You will use the same WebGL code template you used for Exercise 4 and the **Phong** shader you implemented among various BRDFs. For each module you will need to create and implement a new shader program (vertex shader and fragment shader) like you did before.

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 from the drop-down box. Ignore one of the two material property sets, or use the same properties for both. You may export the current rendering as a PNG image file.

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. The template code should run on any recent version of Firefox and Google Chrome web browsers, but does *not* support Safari or Microsoft Internet Explorer.

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.

Slider Controls[1]

The provided UI includes slider controls that may be connected to shader uniforms. You may use them for your convenience—for instance, they can be used to tweak your texture parameters and see their effects interactively. Create a JSON[2] file that contains the specification of the parameters—e.g., the shader uniform name, the range of possible values, the default value, etc. Name it the same as the shader name. Each associated slider will be automatically created and attached to the UI. Missing JSON file means no sliders. Refer to the code template for more details.

[1]The inclusion of slider controls was inspired by Ivo Steinmann, and part of code was taken from him.

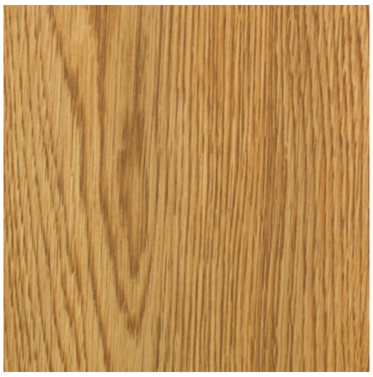
[2]JSON is essentially a subset of JavaScript and used to describe data objects that consist of attribute-value pairs. See for instance this for more information.

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. 1(a).


(a) Wood

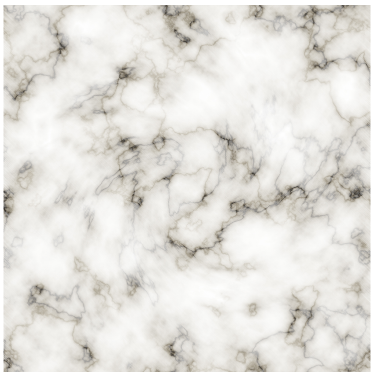

(b) Marble

Figure 1: 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. 1(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[3].

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[4].

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.

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

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.