Computer Graphics - 2014/2015

Exercise 2 - Part C

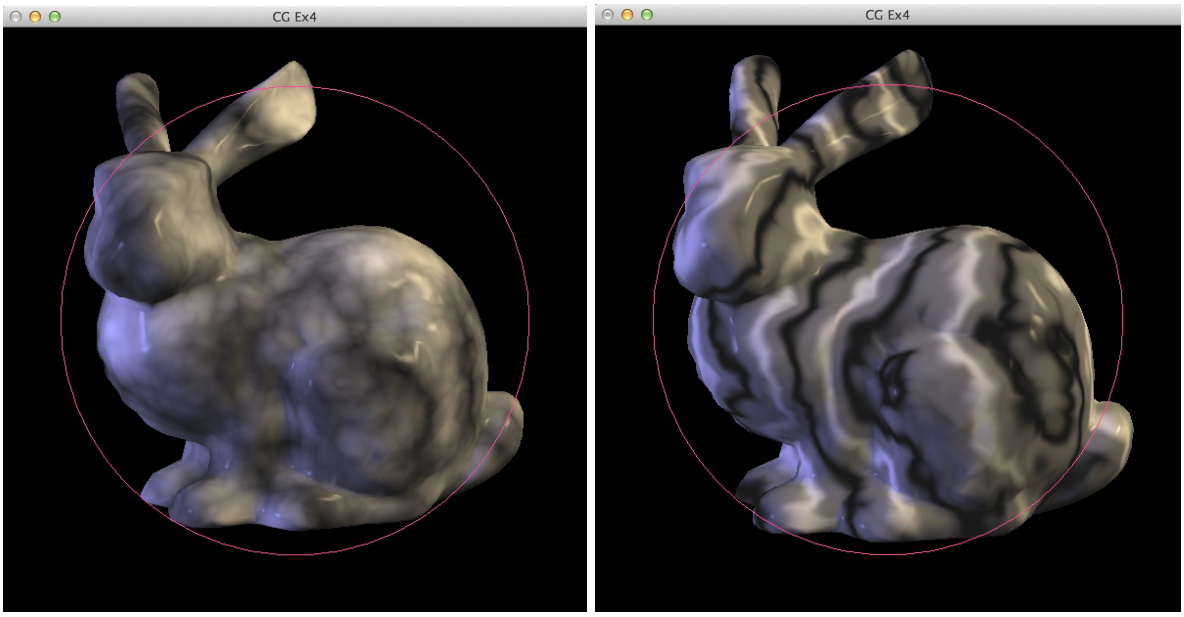
Last update: Dec. 28

In the previous exercises we created an application for interactive viewing of 3D models using different implementations of the Phong lighting model.
In this exercise we will use procedural textures, environment mapping, and bump mapping to make the objects look as if they were made from different materials. Don't let the length of this document intimidate you: you will definitely have to write much less code for this exercise than what you had to write for each of the previous exercises!

Detailed description

Include all functionality defined for the previous exercise. In addition, add the following texturing modes when using the Phong shading model (no need to implement them with Gouraud shading). Your solution does not need to be pixel identical to the school solution; on the contrary, you are encouraged to experiment, and the explanations below describe just a few possible ways among many of achieving the desired appearance.

1. Marble



You should procedurally modify the diffuse reflectance coefficients kd in your fragment shader, so as to approximate the appearance of an object made of marble. You may use the provided turbulence function to achieve this. This function maps 3D points to scalars creating natural

looking turbulent patterns. One possibility is to simply apply this function to each point in the object (after applying a scaling factor to control the frequency of the result). The resulting turbulent field may be used as the diffuse reflection coefficient (after scaling by another parameter). The result is shown above (left image). In this case, the shader code might look something like this:

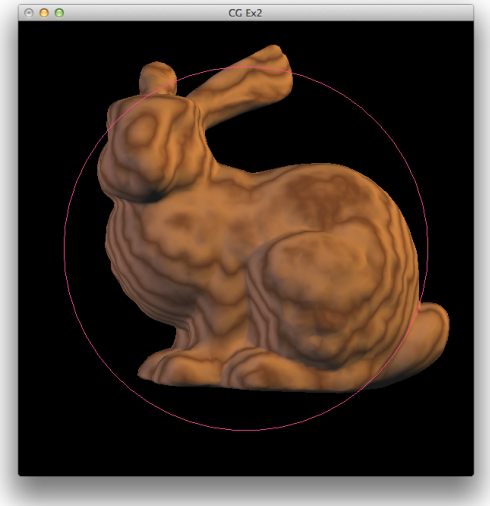
```
kd = turbulenceCoeff * turb(textureScale * texturePosition.xyz);
```

Your task is to generate a somewhat more structured marble appearance, as shown above on the right. One way to do this is to modify the scaled texture position with the turbulence function and then use the result as the argument of a sine function (see the tirgul slides, slide 57). This approximates the deformation and mixing of geological layers with each other over time.

Note that the sine function return values smoothly varying between -1 and 1, so you will need to find a suitable mapping of these values into the [0,1] range. Also, note that this marble texture should only affects the RGB diffuse coefficients. The specular and the ambient coefficients should remain as defined for the previous exercise.

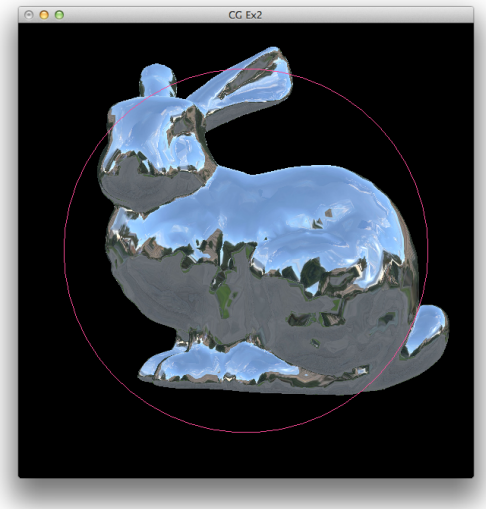
Use the model's bounding box in order to transform each object position into a texture position in the unit cube (similarly to the transformation you had to apply for the RGB cube shading in previous exercises). You should then be able to control the visible density/frequency of the turbulence via the `textureScale` parameter. You should be able to control this parameter interactively using the keyboard (press 's' to reduce the value of this parameter and 'd' to increase it). You should also able to control the magnitude of the turbulence via the `turbulenceCoeff` parameter (press 'f' to reduce the value of this parameter and 'g' to increase it).

2. Wood



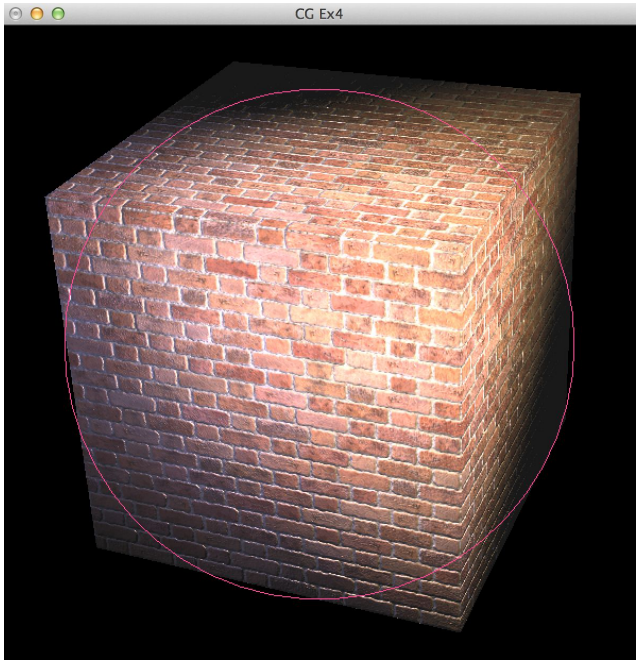
Wood appearance can be approximated by using the turbulence function to perturb concentric cylinders (see the tirgul slides, slide 56). The resulting scalar value may be used to generate transitions (using the GLSL `mix` function) between a darker wood color and a lighter one. The resulting color is used as the diffuse RGB reflectance coefficients, kd. In this case, you should also reduce the specular coefficients, compared to their values for the marble material, but keep the ambient coefficients the same. As in the previous case (marble), interactively control the spacing of the wood veins via the `textureScale` parameter, and the amount of turbulence via the `turbulenceCoeff` parameter (using the same keys as in the previous case).

3. Mirror (environment mapping)



You are given an environment map in spherical coordinates (see tirgul slides 35-36). The x-coordinate of the image maps to values of the longitude angle theta, while the y-coordinate maps to the latitude angle phi. You should display the object as if it was made from an ideal mirror, reflecting the provided environment. You may assume the environment is infinitely far from the object, and therefore the reflection at each point depends on the surface normal at that point, as well as the direction from the point to the camera (see tirgul slide 47). In your host program you should first read the provided environment map and create a texture containing it (reading the BMP image and creating the texture may be done similarly to the example code you were shown in the tirgul). You should then generate the texture coordinates at each fragment and sample the texture on the fly in your fragment shader. Note that the texture coordinates at each fragment are *dynamic*, since they depend on the angle between the normal and the direction to the camera.
Tip: in your shader code you should use the two argument version of the built-in GLSL atan function in order to correctly obtain the polar angles theta and phi.

4. Bumpy bricks



This part only has to work correctly with the cube model! Read the provided brick texture image, and the corresponding bump-map image. Apply the brick texture to each of the faces of the cube, and use the bump map to modify the normal at each fragment, so as to approximate a bumpy appearance. This is done by numerically approximating the two partial derivatives dB/du and dB/dv of the the bump map B at every texture coordinate (u,v). These derivatives (multiplied by the `textureScale` parameter) are then subtracted from the corresponding components of the fragment's normal vector (which then must be renormalized).

Tip 1: the partial derivatives of a digital image B at location (x,y) may be approximated as:
dB/dx = B(x+1,y) - B(x,y)
dB/dy = B(x,y+1) - B(x,y)

Tip 2: since you will be working with multiple textures, you need to associate a separate texture unit with each texture, and make sure that you make each texture unit active before passing the texture data and setting any texture parameters (using glActiveTexture).

Switching between texturing modes:

Switch between texturing modes in a cyclic fashion when the user presses 't' (or 'T'), as done in the school solution. The program should start with the marble texturing mode. Pressing 't' (or 'T')

should then switch to the wood texture mode, followed by the environment mapping mode, followed by the bumpy bricks, and then the “no texture” mode (use the shading parameters defined in the previous exercise), and then back again to the marble texture mode.

Important: Your program should support all features that were defined in the previous exercise.

Keyboard Operations:
R/r : Reset to the initial view of the model: rotation, translation and zoom
Q/q : Quit the program. Simply call exit().
W/w : Toggle between wireframe and full mode.
P/p: Toggle between perspective projection (default) and orthographic projection. You should make sure that the orthographic projection is “compatible” with the current perspective view, i.e. it displays the object at roughly the same size, and responds similarly to zooming in and out.
N/n: Switch between normal estimation modes (i.e. per vertex / per face)
‘1’: Use the full Phong model
‘2’: Use the Gouraud approximation
‘3’: Assign pixel colors as defined in the previous exercise.
‘=’: Increase the shininess coefficient up to 2000.
‘-’: Decrease the shininess coefficient down to 0.
T/t: Switch between texturing modes (see above).
S/s: Decrease the scaling of the texture coordinates (or the scale of the bumps)
D/d: Increase the scaling of the texture coordinates (or the scale of the bumps)
F/f: Decrease turbulence magnitude
G/g: Increase turbulence magnitude

Mouse operations:
Mouse events should be interpreted as defined in the previous exercise (with the appropriate changes to support the new modes defined here).

Bonus (up to 10 points):
Implement “glow” effect as you saw in the TA session:

- Render the scene to an offscreen texture-backed framebuffer and attenuate dark colors (so effectively mostly bright colors will be rendered).
- Blur the result (while rendering to another framebuffer).
- Render the scene again, while blending the blurred image with the actual scene.
- The keys h (or H) and j (or J) should decrease/increase the blur radius (use a gaussian filter) between zero and 50.
- They keys V/v should switch between three modes:
    1. No glow
    2. Only glow
    3. Blended result

Submission

Include the following in your submission

1. A Readme.txt file, that includes:
   - Your id and login
   - Your partner's id and login
   - A brief description of your implementation, which piece(s) of code you started from and what changes you made to it (/them).
   - All files that are required for compilation of your solution with a single 'make' command, and the shaders necessary to run it.

Pack all files as a single zip file named by the following pattern: ex2c_<your 9 digits id>_<your_username>_<your partner's 9 digits id>_<your partner's 9 digits login>.zip (e.g. '`ex2c_123456789_mylogin_987654321_myfriendlogin.zip`').

School Solution:

You can find the school solution here. Before running the school solution, run the set path command as you did in the previous exercise.

Turbulence code and texture images:

Here is the GLSL code that implements the turbulence function: `float turb(vec3 v)` and its dependencies. This code was borrowed from here: http://glsl.heroku.com/e#812.1

Here are the texture images that you need for this exercise. Use the BImage class to load these images into your host program, as was shown in class. You will need to add to your makefile the following linking flags: -L/cs/course/2013/cg/lib -lbimage

Deadline:

You have to submit your solution (via the course's moodle webpage) no later than Tuesday 10/1 at 23:45. Late submission will result in 2^(N+1) points deduction where N is the number of days between the deadline and your submission (rounded up, the minimum grade is 0, friday and saturday are excluded).

Good luck and have fun!