Assignment 5: Environment Mapping with Shaders

Comp175: Computer Graphics – Fall 2022

There's no Algorithm worksheet for this assignment!
Assignment due: **Wednesday December 7** at 11:59pm (Midnight)

1 Introduction

As we have experienced in class and with the lab, shaders is a flexible way for a graphics developer to better express complex mathematical functions. However, setting up shaders is often not a trivial process, considering that the developer will need to coordinate the memory management and data passing between the CPU and the GPU.

In this assignment, you will be implementing "spherical" environment mapping. The idea behind environment mapping is to capture an environment as a texture map and have a (shiny) object in the scene "reflect" the environment in real time by applying the texture.

The learning goals of this assignment are to:

- Implement a spherical environment map with shaders.
- Learn to manage shaders. In particular, learn to apply different shaders to different objects in the scene.
- Work with multiple textures. Further, combine the textures with the lighting equations from your ray tracer (i.e. diffuse shading).

2 Requirements

We list the major features that you have to implement for this assignment.

2.1 Multiple Objects with Different Shaders

There should be two objects in your scene: (1) `environment sphere:` a large sphere that represents the environment, and (2) `reflective object:` the object that the environment is mapped on to.

These two objects should have different associated shader files. Note: you are **NOT allowed** to have only a single set of shader files (one vertex and one fragment) and use an if or switch statement for rendering the two different objects. To be clear, each of the two objects should have its own vertex shader and fragment shader. At run time, your program should switch the use of the different shader file(s) when rendering the two objects.

2.2 Spherical Environment Map

You are required to implement spherical environment mapping. The high-level concept of this is similar to that of ray casting – from the eye point, you would cast a ray into the world. Should the ray intersect with the `reflective object` (at `isectPoint1`), the reflective ray will bounce off the object and hit the larger `environment sphere` (at `isectPoint2`). This large `environment sphere` should be texture-mapped with some image (in this assignment we are using images that have been captured with 360 degree spherical cameras). The texture color at `isectPoint2` should be used as part of the calculation of the color of the pixel / fragment (in addition to the multiple textures and the diffused shading mentioned below).

Your code should work with any arbitrary object, including a sphere. This should be at no additional cost to you – if you implement your spherical mapping correctly, the same shader code should work with all other objects.

Be careful with the position and orientation of the texture maps. (1) Text in the environment should be shown in "reverse" in the `reflective object`; (2) The relative positions of the two texture maps should make sense physically. For example, text that is visible in the `reflective object` should be located behind the camera's position.

2.3 Multiple Textures and Diffuse Shading

In addition to the environment (texture) map, your `reflective object` should also have its own texture

as well that can be changed at run time. Your interface comes with a slider that allows the user to choose the blending between the environment map and the object's own texture color. Finally, you should implement diffuse shading to give the object its proper shading.

2.4 Supporting the User Interactions

Your program should support all the interactions that are a part of the user interface. In particular, there are five components to the user interface:
Environment: Your program should be able to load different environment maps (as PPM files) at run time. Note that some of the images in the `data` folder are quite large and can take a few seconds to load.

Object Model: These sliders control the rotation and scale of the `reflective object`. Note that changing these values should not affect the `environment sphere`.

World Model: These sliders control the rotation and scale of the camera / world.

Object: You should be able to load a different PLY file or a different texture for the `reflective object`. The `Texture Blend` slider should control the blending between the environment map and the texture map (where 0.0 means 0% from the texture map (and 100% from the environment map) and 1.0 means the reverse). Lastly, the checkbox `Diffuse Shading` turns on and off diffuse shading. There is only one light source in the scene. Its position is "hard coded" in `MyGLCanvas`.

Shader: The button allows you to re-compile your shader code. You shouldn't have to do anything here.

3 Support Code

The compiling of shaders and textures is already done for you. For this assignment, there are only three things that you need to do:
Variable Passing: In `MyGLCanvas` you need to specify and set up the variables that you want to pass to the (two different sets of) shaders. Look out for the `TODO` comments in the code.

Shaders for Reflective Object: You need to implement spherical environment mapping by editing the two shader files (vertex and fragment) for the `reflective object`.
Shaders for Environment Sphere: You need to implement a textured sphere by editing the two shader files (vertex and fragment) for the `environment sphere`. Note that you should manually scale a default unit sphere in the shader to enlarge it (for me, my scale factor is 7).

4 Demo

Normally, your shader code is stored as text files (one for vertex shader and one for fragment shader) that can be parsed and executed in run time. As you have experienced in your first shader lab, this allows you to "play" with shaders without the need to recompile the entire C++ program.

For the demo, we cannot include the shader code as part of the demo as text files. Doing so will effectively be giving away the solution (not that we don't trust you to do your own work, but... we don't). Instead, we have "hard coded" the shader code as a C++ string that will be included as part of the compilation.

In this regard, the demo will be slightly different from what you build.

5 How to Submit

You will be working in a team of 3 people. You are encouraged to discuss your strategies with your classmates. However, each team must turn in ORIGINAL WORK, and any collaboration or references outside of your team must be cited appropriately in the header of your submission.

Submit all files (not just the files you changed) as a zip file via Canvas. Remember, during in-person grading, we will be downloading your submission from Canvas and asking you to compile it on your computer in real time. In order to maintain fairness, you will not be allowed to alter your submitted zip file at that point.