Foundations of Computer Graphics

Assignment 3

Instructions:

- Assignments can be submitted in groups of at most three. The purpose of groups is to learn from each other, not to divide work. Each member should participate in solving the problems and have a complete understanding of the solutions submitted.
- Submit your assignments as a zip file (one per group) which includes the Common directory and a separate directory for each of the assignments so that we can run your code by just extracting the zip file and double-clicking the html files.

Note: This assignment is worth 30 points but it is possible to earn upto 50 points due to bonus points. Excess points over 30 will also count towards your overall score in the course.

Problem 1 (10 points)**.**
Implement a sky box, using a cube along with `cubemap.jpg` as the texture. The camera should be placed at the center of the cube. Since the texture represents scenery at a distance the camera location should always be at the center of the cube but the user should be able to change the gaze direction using the arrow keys. Only diffuse lighting should be used.

Add a teapot (`teapot.obj`) to the scene and implement ***reflection mapping*** [1] on the teapot.

It should be possible to move the camera in the scene and when it is translated, the teapot should move relative to the camera but the skybox shouldn't since it represents scenery at a distance. This can be achieved by making the sky box cube very large but that is very inefficient. An efficient way is to keep the cube small and always move it so that its center is at the camera location. To avoid the cube from obstructing the view, you can disable writing into the depth buffer when drawing the cube. Effectively then it appears that the cube is very large. To disable writing into the depth buffer, you can use `gl.depthMask(false)` and to enable it again use `gl.depthMask(true)`.

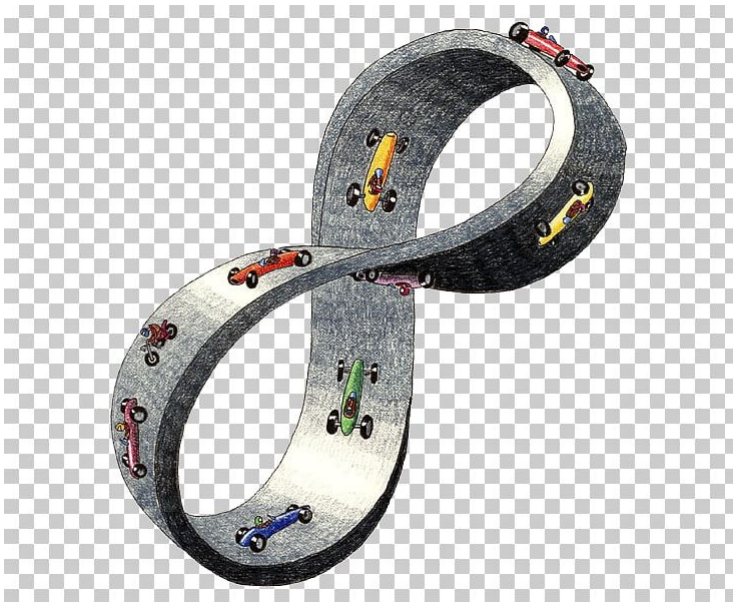
Bonus (5 points). Replace the sky box texture above by `cubemap1.jpg`. Replace the 'grass field' in the texture by a terrain by creating a large square grid on the xz-plane (that occludes the grass field) and giving a height (y-coordinate) to each grid point using a 2D noise function. The height added should not be very large - we don't want hills or mountains, just a more natural landscape. You don't need to update the reflection mapping based on the terrain.

Apply the texture `moss-diffuse.jpg` and the normal map `moss-normal.jpg` on the terrain. In order to avoid streching the images, you need repeat the texture and normal maps over the surface. For instance, the texture coordinates of a point (x, y, z) can be set to $(x - \lfloor x \rfloor, z - \lfloor z \rfloor)$. Set the shininess constant in Phong Lighting to a small value since we don't expect the terrain to be very reflective. The user should be able to move around the scene. As before the sky should not move relative to the user but the ground should.

[1] See slide 15 of Lecture 12 or the wikipedia article `https://en.wikipedia.org/wiki/Reflection_mapping`.

Problem 2 (10 points)**.**
Implement a scene where a car is moving on a Möbius strip as shown in the picture below. You can choose the shape of the strip as per your convenience. The file `formula1.obj` contains a model of a Formula 1 car which you can use. The texture and normal maps for the car are in `car_diffuse.png` and `car_normal.png` respectively. Use repititions of `road_texture.jpg` for the texture on the road.

Bonus (5 points). The file `car_shininess.png` contains a grayscale image in which the intensity at a texture coordinate indicates the shininess value to be used at that point. You can scale any of the r, g or b components of the color by a large number (e.g. 100) to obtain the shininess coefficient. Use this to vary the shininess constant in the Phong lighting from fragment to fragment.



Bonus (5 points). Allow the user to switch to a view from the driver's seat. The camera should be looking straight ahead towards the road.

Problem 3 (10 points)**.**
Modify the ray tracing program so that most of the computation is done in the fragment shader. The idea is to draw a square that covers the screen. Each pixel corresponds to a fragment and the corresponding fragment shader can compute the color of the pixel by ray tracing. You need to move the entire scene set up and the relevant functions to the fragment shader.

Since GLSL ES does not support recursion, you will need to use the non-recursive implementation of the basic ray tracer we have implemented (see the code on `Github`). You may need to use C-style structs in the fragment shader. GLSL ES does not support C++ style classes. Please have a look at `http://math.hws.edu/eck/cs424/notes2013/19_GLSL.html` to learn how to use these.

Further modify the code to add ***anti-aliasing***. This can be done by shooting multiple random rays through a pixel and taking the average of computed colors.

Your program should display an animated scene with at least two lights and two objects.

Bonus (5 points): Display a billiard ball by putting the texture `billiard_ball_texture.jpg` on a sphere. It should be possible to rotate the billiard ball about its center using the virtual trackball interface. The ball should look similar to the ball shown in `billiard_ball.jpg` (ignoring the reflection of the area lights in the image).