CPSC 314
Assignment 4: Textures and Shadows

Due 11:59PM, April 3, 2024

1 Introduction

In this assignment you will be learning about different uses of textures. You will implement a skybox, give your favourite armadillo a new, dazzling look, and make it cast shadows. And you will meet a new protagonist, Shay D. Pixel, a SIGGRAPH mascot, explore physically based rendering further and have a taste of image-based lighting.

1.1 Getting the Code

Assignment code is hosted on the UBC Students GitHub. To retrieve it onto your local machine navigate to the folder on your machine where you intend to keep your assignment code, and run the following command from the terminal or command line:

```
git clone https://github.students.cs.ubc.ca/CPSC314-2023W-T2/a4-release.git
```

1.2 Template

- The file `A4.html` is the launcher of the assignment. Open it in your preferred browser to run the assignment, to get started.
- The file `A4.js` contains the JavaScript code used to set up the scene and the rendering environment.
- The folder `glsl/` contains the vertex and fragment shaders.
- The folder `js/` contains the required JavaScript libraries. You do not need to change anything here.
- The folder `gltf/` contains geometric models to be used in the scene.
- The folder `images/` contains the texture images used.

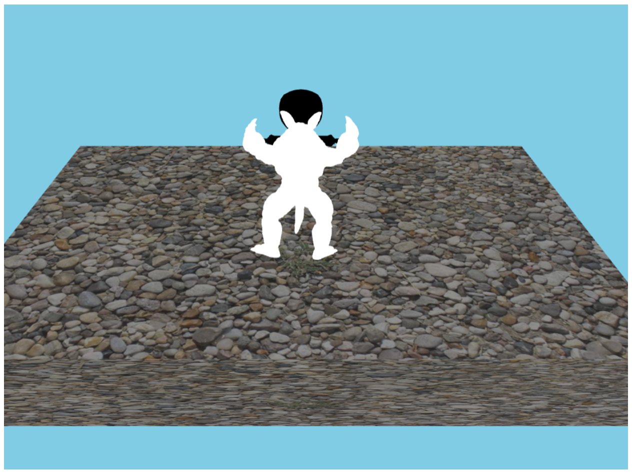

Figure 1: Initial configuration.

2 Work to be done (100 points)

Here we assume you already have a working development environment which allows you to run your code from a local server. If you do not, check out instructions from Assignment 1 for details. The initial scene should look as in Fig. 1.

2.1 Part 1: Required Features

a. **(15 points)** Texture Mapping with ShaderMaterial.

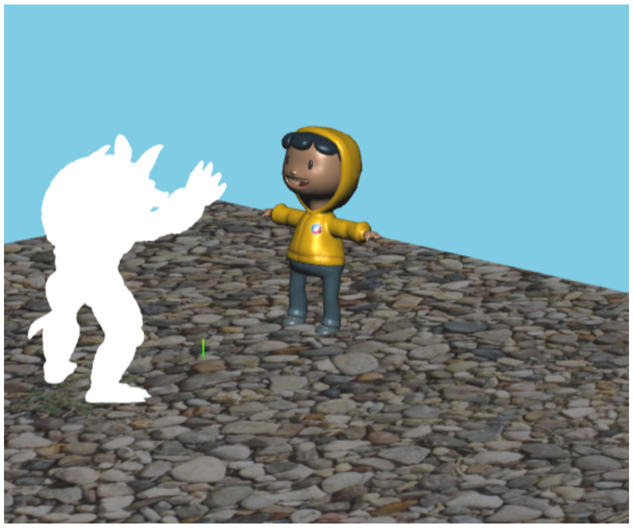

Figure 2: Question a: Texture mapping with ShaderMaterial.

In this part you will implement texture mapping for Shay D. Pixel using shaders. You are provided with color texture `images/Pixel_Model_BaseColor.jpg`. The geometric model `gltf/pixel_v4.glb` is in GLB format which you have seen in previous assignments, and it has vertex UV coordinates baked in.

Your task is to pass the textures (as a uniform) to the fragment shader (`shay.fs.glsl`), use the right UV coordinates to sample a color from the texture, and then use the sampled color and the light intensity (which has been computed for you in `shay.fs.glsl`) to calculate the final fragment color. The result should look close to what is shown in Fig. 2.

Hint 1: The texture is flipped on the y-axis. Take this into consideration when you assign the UV coordinates.

b. **(15 points)** Skybox.

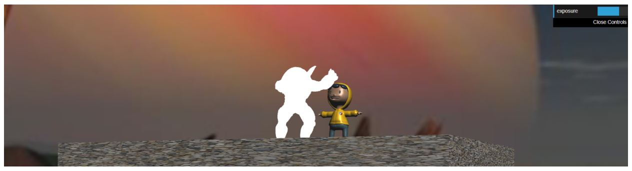

Figure 3: Question b. Skybox.

A skybox is a simple way of creating backgrounds for your scene with textures. We have provided six textures under `images/cubemap/`. You will implement a skybox using cube environment mapping as discussed in class. Specifically, in `A4.js` load the six textures to `skyboxCubemap` in a proper order; then make changes to the shaders (`skybox.vs.glsl` and `skybox.fs.glsl`). In Three.js you can add a skybox background by setting `scene.background` to `CubeTexture`: see `https://threejs.org/manual/?#en/backgrounds`. The final result should look close to what is shown in Fig. 3.

Hint 1: Remember from lectures that you need to define a direction vector to sample a color from the cubemap. You can define it in the world frame; also think about how to make use of the fact that the cube which the texture is mapped to is centered at the origin.

Hint 2: Offset your pixel vertex position by the cameraPostion (given to you in world space) so that the cube is always in front of the camera even when zooming in and out.

c. **(20 points)** Shiny Armadillo.

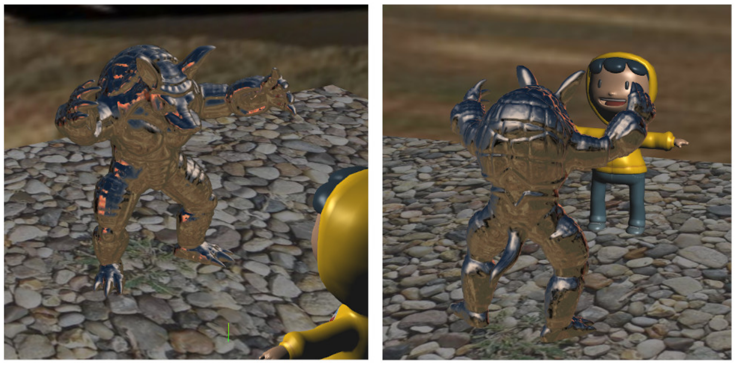

Figure 4: Question c. Shiny Armadillo.

Another interesting use of `samplerCubes` is **environment mapping**. This can be used to make the same, boring armadillo highly reflective, like a mirror. For this part, complete the shaders `envmap.vs.glsl` and `envmap.fs.glsl` to implement a basic reflective environment map shader. The result should be close to what is shown in Fig. 4. You can use the same cube texture `skyboxCubemap` in your shaders which is passed as a `samplerCube` uniform like before, as well as the same `texture()` function, but pay attention to use the correct `vec3`, described in class and in the textbook, to retrieve the texture color.

Hint 1: Try replacing the armadillo with a sphere object to start off with so it is easier to inspect and debug your environment map.
Hint 2: Think about how the armadillo (or sphere) should look from various directions. How should it look from the bottom? The top?

d. **(40 points)** Shadow Mapping.

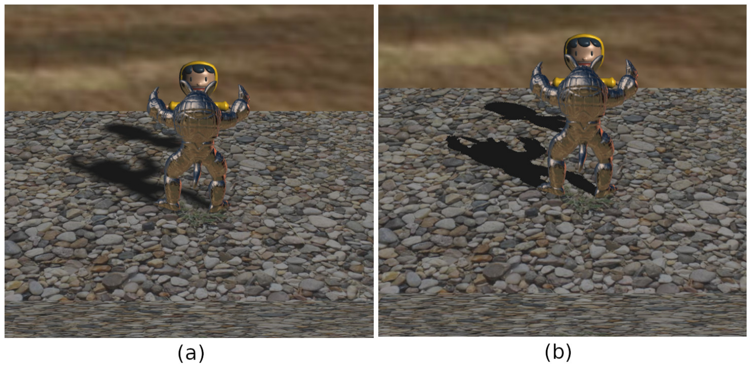

Figure 5: Question d: (a) Smoothed shadow mapping. (b) Non-smoothed shadow mapping.

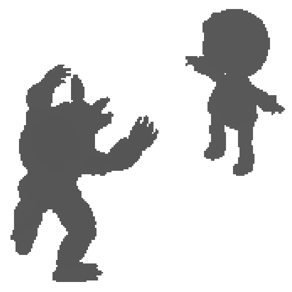

Figure 6: Question d: Depth Map.

Shadows are a tricky part of computer graphics, since it is not easy to figure out which parts of a scene should be cast in shadow. There are many techniques to create shadows (raytracing, shadow volumes, etc.); we will use shadow mapping in this assignment. Shadow mapping is all about exploiting the z-buffer: a shadow map is rendered in an off-screen frame buffer by projecting the scene from the perspective of a light source, giving us a depth-like value at each fragment along rays of the light source. Your task is implementing shadow mapping, so that smooth shadows of the armadillo and Shay can be casted onto the floor: see Fig. 5 (a). You can switch between scene views by pressing key 1, 2, and 3 for the scene, depth scene, and shadowed scene respectively. Scene 2 and 3 are for you to implement; the result from rendering scene 2 is shown in Fig. 6. We have listed the steps for you to follow:

1. Start by creating the shadow map, which is the depth map when viewed from the camera. Add appropriate object(s) to the provided `shadowScene`, do a first pass render to a `WebGLRenderTarget` to create the depth map, and finally visualise this using the provided `postScene` (short for "post processing scene"). Your primary job will be to pass the appropriate textures between render targets and to implement the render shaders (`render.vs.glsl` and `render.fs.glsl`). You will find the API docs useful: `https://threejs.org/docs/#api/en/renderers/WebGLRenderTarget`.
2. Next, use the depth map to project the shadows onto the floor. This will involve modifying the floor's shader code to check whether a fragment is in shadow or outside. You can do this by transforming the fragment's position to "light space" (i.e., in the shadow camera's coordinate frame), and using that to compute the appropriate texture coordinate in the shadow map. Then compare the depth of the fragment to the value stored in the shadow map. After this step you should see casted shadowns similar to ones shown in Fig. 5 (b).

3. Lastly, you will smooth the shadows by using percentage closer filtering (PCF), in the floor shader code. PCF is a shadow anti-aliasing technique which reduces 'jaggies' by replacing the binary in/not-in shadow calculation of a pixel, with a calculation that instead checks if a pixel and its neighbours are in shadow, and 'shadows' the pixel according to the fraction that are.

e. **(10 points)** Image-based Lighting.

Image-based lighting (IBL) is a rendering technique which requires capturing real-world light information as a High Dynamic Range (HDR) image. This image is then used to calculate the lighting for objects in the scene. IBL can create incredibly realistic simulations of real world lighting, especially when used with physically based materials as we discussed in class. You will set up a scene in three.js to use IBL. You can enter the IBL scene by pressing key 4.

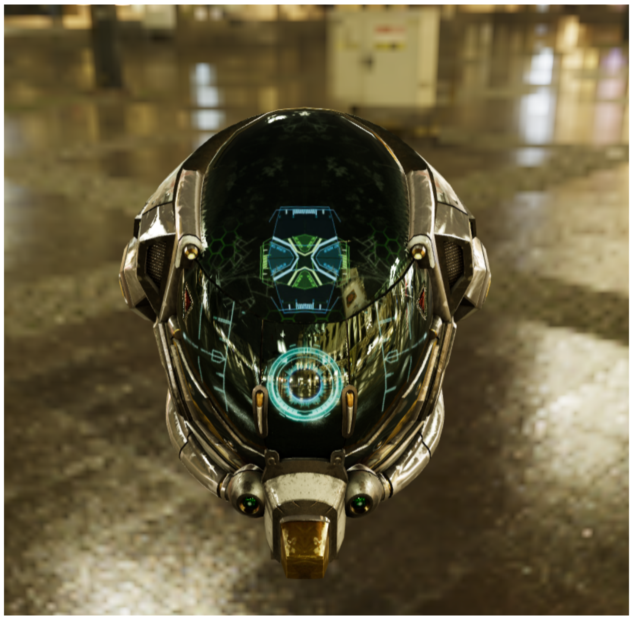

Figure 7: Question e: Completed IBL scene.

Even though this can be complicated, we have made the problem as simple as possible, so that you can get a flavor for how to use these techniques without getting bogged down. In A3 you have already learned how to load a scifi-themed helmet and apply PBR when texturing it. For this assignment you just need to complete `helmetMaterial` by setting the material's environment map property after loading the HDR texture. The result should resemble what is shown in Fig. 7.

2.2 Part 2: Creative License (Optional)

You have many opportunities to unleash your creativity in computer graphics! In this **optional** section, and you are invited to extend the assignment in fun and creative ways. We'll highlight some of the best work in class. A number of exceptional contributions may be awarded bonus points. Some possible suggestions might be:

- Experiment with multi-pass rendering to create a texture out of the current scene to use for the armadillo environment map, such that the armadillo appears to reflect the objects in its surroundings.
- Experiment with other graphics techniques, like procedural mapping (eg. Perlin noise), subsurface scattering ("Gaussian blur"), ambient occlusion, and raytracing.
- Make a short film/animation and tell a tear-jerking story with sounds.
- Make an interactive video game.

3 Submission Instructions

3.1 Directory Structure

Under the root directory of your assignment, create two subdirectories named "part1" and "part2", and put all the source files, your makefile, and everything else required to run each part in the respective folder. Do not create more sub-directories than the ones already provided.
You must also write a clear README.txt file that includes your name, student number, and CWL username, instructions on how to use the program (keyboard actions, etc.) and any information you would like to pass on to the marker. Place README.txt under the root directory of your assigment.

3.2 Submission Methods

Please compress everything under the root directory of your assignment into `a4.zip` and submit it on Canvas. You can make multiple submissions, but we will grade only the last one.

4 Grading

4.1 Point Allocation

Part 1 has 100 points in total; the points are warranted holistically, based on

- The functional correctness of your program, i.e. how visually close your results are to expected results;
- The algorithmic correctness of your program;
- Your answers to TAs' questions during F2F grading.

Part 2 is optional and you can get bonus points (0-10 points) at the instruction team's discretion. The max score for each assignment is 110 points.

4.2 Face-to-face (F2F) Grading

For each assignment, you are required to meet face-to-face with a TA during or outside lab hours to demonstrate that you understand how your program works. To schedule that meeting, we will provide you with an online sign-up sheet. Details regarding when and how to sign up will be announced on Canvas and on Piazza.

4.3 Penalties

Aside from penalties from incorrect solution or plagiarism, we may apply the following penalties to each assignment:

a. **Late penalty**. You are entitled up to three grace (calendar) days in total throughout the term. No penalties would be applied for using them. However once you have used up the grace days, a deduction of 10 points would be applied to each extra late day. Note that

   (a) The three grace days are given for all assignments, **not per assignment**, so please use them wisely;

   (b) We check the time of your last submission to determine if you are late or not;

   (c) We do not consider Part 1 and Part 2 submissions separately. Say if you submitted Part 1 on time but updated your submission for Part 2 one day after the deadline, that counts one late day.

b. **No-show penalty.** Please sign up for a grading slot at least one day before F2F grading starts, and show up to your slot on time. So a 10-point deduction would be applied to each of the following circumstances:

   (a) Not signing up a grading slot before the sign-up period closes;

   (b) Not showing up at your grading slot.

   If none of the provided slots work for you, please contact the course staff on Piazza before the sign-up closes. Also, please note that

   (a) we would not apply the no-show penalty if you are unable to sign up/show up on time because of a personal health emergency, and in such cases we would like to see a written proof of the situation.

   (b) if you decide to make a late submission and only get graded after the F2F grading period ends, you need to show the TA your submission time on Canvas so he/she knows you submitted late and will not apply the no-show penalty.

   (c) In the past some students reported that they got their names overwritten by others, or their names disappeared mysteriously due to technical glitches. Therefore we suggest you to take a screenshot of your slot after sign-up just to prove you have done it.