CPSC 314
Assignment 4: Textures and Shadows

Due 11:59PM, April 2, 2025

1 Introduction

In this assignment you will be learning about different uses of textures. You will implement a skybox, give Keenan Crane's Cow a couple of new, dazzling looks, and make it cast shadows. And you will meet a new protagonist, Shay D. Pixel, a SIGGRAPH mascot!

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/CPSC314-2024W-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.

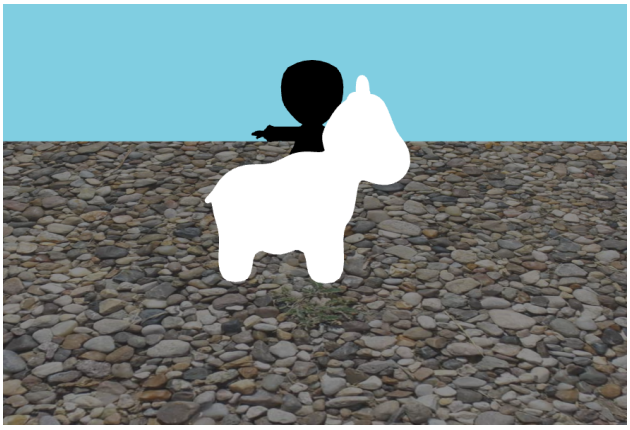

Figure 1: Initial configuration.

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. **(10 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. **(10 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`). 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 Cow.

Figure 4: Question c. Shiny Cow.

Another interesting use of `samplerCubes` is **environment mapping**. This can be used to make our cow highly reflective, like a mirror. For this part, complete the shader `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 cow with a sphere or cube object to start off with so it is easier to inspect and debug your environment map.
Hint 2: Think about how the cow (or sphere) should look from various directions. How should it look from the bottom? The top?

d. **(35 points)** Shadow Mapping.

(a) Left Image (b) Right Image

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 cow 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)** Procedural Textures and Noise

Instead of using predefined textures, you can also create your own from scratch! In this part of the assignment, you will use a hashing function to create **perlin noise**. Perlin Noise is a pseudo random noise where adjacent inputs are likely to have similar outputs resulting in more wave-like, undulating patterns. You can learn more about them here.

Figure 7: Question e: Before Spots

Figure 8: Question e: After Spots

(a) Firstly, the cow needs to be rendered similarly to Shay in part a using shaders 'cow.vs.glsl' and 'cow.fs.glsl'. Once the cow is rendered it should look like Fig. 7.

(b) Next, to put a texture on a cow, use perlin noise in conjunction with a step function on the UV coordinates of the texture map of the cow to give it its cow-patches.

(c) Finally, take a look at the texture map for the cow `images/SpotTexture.png` to set some boundary on where not to apply the noise the stop the cow patches from appearing on the face. The final scene should look like Fig. 8.

f. **15 pts** Feature Extension.

In this part, you are required to extend the assignment to add a feature of your own choosing. The goal is to encourage you to explore the capabilities of Three.js and WebGL. For full credit for this part, the feature does not have to be complex or creative, but must be non-trivial (e.g., not just changing a color). Roughly requiring about 10 lines of new code.

For this question, first duplicate your current work into a new directory named 'part2', then implement your feature in this new directory. Write a brief description of your feature in the README file. You will be graded on both how it works and how well you can explain your feature.

3 Bonus Points

You have many opportunities to unleash your creativity in computer graphics! In particular, if you create a particularly novel and unique feature extension, you may be awarded bonus points. A small number of exceptional extensions may be shown in class, with the student's permission.

4 Submission Instructions

4.1 Directory Structure

Your submission should contain two subdirectories - the first should be named 'part1' and should contain all parts before the feature extension; the second subdirectory should be named 'part2' and should contain your feature extension. For each of the two subdirectories, include all the source files and everything else (e.g. assets) needed so each part can be run independently. Do not create more sub-directories than the ones already provided.
You must also includes your name, student number, CWL username, and instructions on how to use the program (keyboard actions, etc.) for your feature implementation, and any extra information you would like to pass on to the marker in the provided `README.md` file.

4.2 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.

5 Grading

5.1 Face-to-face (F2F) Grading

Submitting the assignment is not the end of the game; to get a grade for the assignment, you must meet face-to-face with a TA in an 8-min slot during or outside lab hours, on Zoom, to demonstrate that you understand how your program works. To schedule that meeting, we will provide you with an online sign-up sheet. Grading slots and instructions on how to sign up will be announced on Canvas and on Piazza. During the meeting, the TA will (1) ask you to run your code and inspect the correctness of the program; (2) ask you to explain parts of your code; (3) ask you some questions about the assignment, and you will need to answer them in a limited timeframe. The questions will mostly be based on ThreeJS or WebGL concepts that you must have come across while working on the assignment; but also you may get conceptual questions based on lecture materials that are relevant to the assignment, or technicalities that you may not have thought about unless you really "dug deep" into the assignment. But no need to be nerveous! We evaluate your response based mainly if not entirely on the coherence of your thoughts, rather than how complete or long your response is: e.g. if the full answer to a question includes A+B+C+D and you mentioned A+B only in the provided time, but your thread of thought is logical then you may still get full marks.

5.2 Point Allocation

All questions before Feature Extension has a total of 85 points. The points are warranted 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, e.g. applying transformation matrices in the right order;
- Your answers to TAs' questions during face-to-face (F2F) grading.

The Feature Extension component is valued at 15 points. You will earn some points as long as you implement at least one extension that the grading TA considers novel and unique. However, full marks may not be awarded if your implementation introduces significant visual artifacts that diminish the product's appeal, or includes critical bugs that render the system unusable (e.g., failing to respond to user commands). Exceptional feature extensions may also be eligible for bonus marks, allowing for a score exceeding 100 on an assignment. However, note that your overall course grade will be capped at 100.

5.3 Penalties

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

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

1. The three grace days are given for all assignments, **not per assignment**, so please use them wisely;
2. We check the time of your last submission to determine if you are late or not.

No-show penalty. Please sign up for a grading slot on the provided sign-up spreadsheet (link will be posted later on Piazza) before the submission deadline of the assignment, and show up to your slot on time. A 10-point deduction would be applied to each of the following circumstances:

1. Not signing up a grading slot before the sign-up period closes. Unless otherwise stated, the period closes at the same time as the submission deadline.
2. Not showing up at your grading slot.

If none of the provided slots work for you, or if you have already missed your slot, follow instructions outlined in this Piazza post to get graded after the F2F grading period ends. Also, please note that

1. you'll need to explain to your TA why you're getting graded late, and may be asked to present documents to justify your hardship. The TA may remove the no-show penalty as long as he/she deems the justification to be reasonable.
2. In the past some students reported that their names disappeared mysteriously due to technical glitches, and the spreadsheet's edit history has no trace of it. So double check that your name is on the sign-up sheet after you sign up by refreshing the page.