UTD CS 4361 Assignment 5: Textures and Shadows

Due on 11:59pm on Dec 02, 2025

1 Introduction

In this assignment you will be learning about different uses of textures. You will implement a skybox, give the provided bunny 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 released on the E-learning page. Please download it to your local machine, navigate to the folder where you intend to keep your assignment code, and run the following command from the terminal or command line:

```
unzip a5-release.zip
```

1.2 Template

- The file `A5.html` is the launcher of the assignment. Open it in your preferred browser to run the assignment, to get started.
- The file `A5.js` contains the JavaScript code used to set up the scene and the rendering environment. You may need to modify it.
- The folder `glsl/` contains vertex and fragment shaders for the bunny and other geometry. You may need to modify shader files in there.
- The folder `js/` contains the required JavaScript libraries. Generally, you do not need to change anything here unless explicitly asked.
- The folder `gltf/` contains geometric model(s) we will use for the assignment, as well as texture images to be applied on the model(s).
- The folder `images/` contains the texture images used.

1.3 Anti-AI/Plagiarism Measures

To prevent plagiarism, 10 students will be randomly selected. These students are required to present their final submitted code to TAs during office hours and point out the locations of core code for the corresponding problems.

2 Work to be done (100 points)

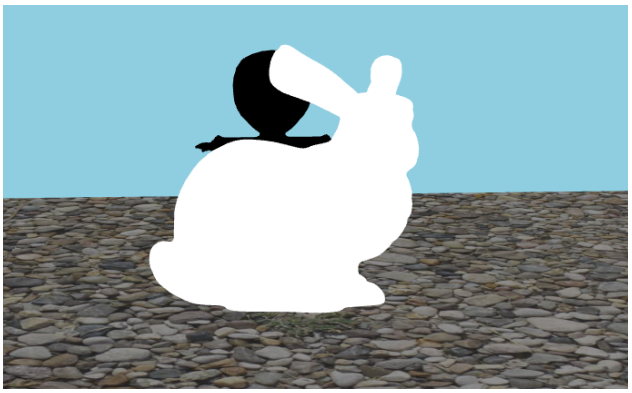

Figure 1: Initial configuration.

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. **(20 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. **(20 points)** Skybox.

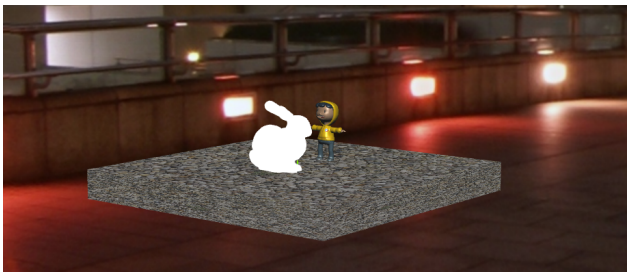

Figure 3: Question b. Skybox.

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 `A5.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 Bunny.

Figure 4: Question c. Shiny Bunny.

Another interesting use of `samplerCubes` is **environment mapping**. This can be used to boring bunny 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 bunny 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 bunny (or sphere) should look from various directions. How should it look from the bottom? The top?
Hint 3: Since cube maps follow a left-handed coordinate system and three.js uses a right-handed coordinate system, you would need to flip the reflected ray in the x direction to get the correct result.

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

Figure 5: Question d: (a) Non-Smoothed shadow mapping. (b) 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 Bunny and Shay can be casted onto the floor: see Fig. 5. 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.

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 bunny environment map, such that the bunny 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

You must submit your final code and write a clear `README` file which includes your name, student number, and the core idea/changed code/explanation/screenshot of each question. The TAs strongly encourage you to read the template of answers in template.md and use a similar template to submit your answers.
Your README file can be in PDF, Word, or Markdown format. It must be placed in the root directory and will serve as the basis for grading your work. **Missing a README file will result in a 30-point deduction.**

3.2 Submission Methods

Please compress everything under the root directory of your assignment into `a5.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

Each assignment has 100 points for Part 1. 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. **Missing a README file will result in a 30-point deduction. Please make sure the README file includes the screenshot of each question.**
TAs will carefully review your README file and evaluate based on idea/code/explanation/screenshot. If the screenshots show correct rendering effects, full points will be graded.

4.2 Anti-AI/Plagiarism Measures

To prevent plagiarism/AI cheating, 10 students will be randomly selected. These students are required to present their final submitted code to TAs during office hours and point out the locations of core code for the corresponding problems. TAs will typically notify selected students via email with meeting links, offline addresses, and a schedule for students to book appointments. Each student will have 5 minutes to answer questions about their code.

4.3 Penalties

Aside from penalties from incorrect solution or plagiarism, we may apply the following penalties to each assignment:
Late penalty. A deduction of 10 points will be applied for each late day. Note that

1. We check the time of your last submission to determine if you are late or not;
2. 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.

AI/Plagiarism penalty. If a student cannot point out where their code is located for solving specific problems (e.g., which lines of GLSL code correspond to the answer for a particular problem) and cannot clearly articulate their approach to solving the problem, this will result in a complete deduction of points for the corresponding assignment.