CPSC 314
Assignment 4: Textures and Shadows

Due 11:59PM, March 29th, 2021

1 Introduction

In this assignment, you will be learning all about textures. You will implement a skybox, a textured floor, as well as give your favourite armadillo a new, dazzling look. The terrain will use basic texture mapping (for color) and and shadow mapping. And you'll 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/cpsc314-2020w-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 the geometric models loaded in the scene.
- The folder `images` contains the texture images used.

2 Work to be done (100 points)

First, ensure that you can run the template code in your browser. See instructions in Assignment 1. The initial scene should look as in Figure 1.

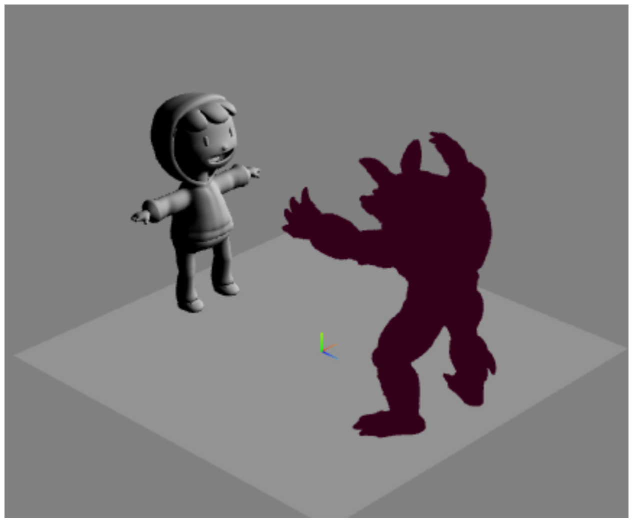

Figure 1: Initial configuration

1. **Part 1: Required Features**

   (a) **(10 points)** Basic Texture Mapping.

   A modeled object often has many very small details and facets of the surface (e.g. the grain of wood on a box, scratches on metal, freckles on skin). These are very difficult, if not impossible, to model as a single material lit by a Phong-like model. In order to efficiently simulate these materials we usually use texture mapping. In basic texture mapping, UV coordinates are stored as vertex attributes in the vertex buffer (Three.js provides them in the `vec2 uv` attribute for default geometries such as planes and spheres). UV coordinates allow you to look up sampled data, such as colors or normals, stored in a texture image, as discussed in class.

   Although our terrain is a simple plane, we can perturb surface normals with a texture and use standard lighting methods to simulate the effect of bumps along the surface. A common way of doing this is with a **normal map** that directly stores modified normals on each point in the surface.

   - For this question, you can use the built-in Three.js materials for texture mapping. You are provided with `images/color.jpg` and `images/normal.jpg` that you have to apply to the square "terrain" that our characters are standing on. You do not need to write a new shader. Your task is to use the built-in Three.js `MeshPhongMaterial` and provide the correct textures to it.

- Once you have completed this step, you should see the floor textured, as shown in Figure 2.

Textures in GLSL are specified as `sampler2D` uniforms, and the values can be looked up using the `texture()` function. Three.js built-in materials do this for you.

In this simple case, you can think of the color map as the diffuse reflectance of the surface.

Hint 1: We load the relevant textures using `THREE.TextureLoader`. You can consult the documentation for more information.

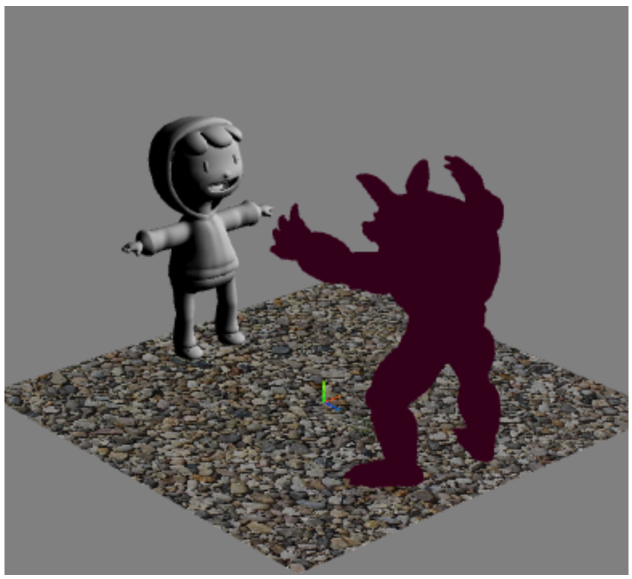

Figure 2: Question 1a) Basic texture mapping

(b) **(30 points)** Texture Mapping with ShaderMaterial.

In this part you will implement texture mapping for Shay D. Pixel using shaders. You are provided with a color texture `images/Pixel_Model_BaseColor.jpg`, and the geometric model has vertex UV coordinates baked in. The model is read from a `glTF` file, a new format that is growing in popularity. We have provided the loader so you don't have to know details of how glTF works.

Your tasks are:

- Complete the `shay.vs.glsl` and `shay.fs.glsl` shaders.
- First shade the Pixel model using the Blinn-Phong model from assignment 3. The diffuse component has already been calculated for you.
- Pass the textures (as a uniform) to the fragment shader and use the right UV coordinates to sample a color from the texture. Use this sampled color and the light intensity from the Blinn-Phong model to calculate the final fragment color.

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

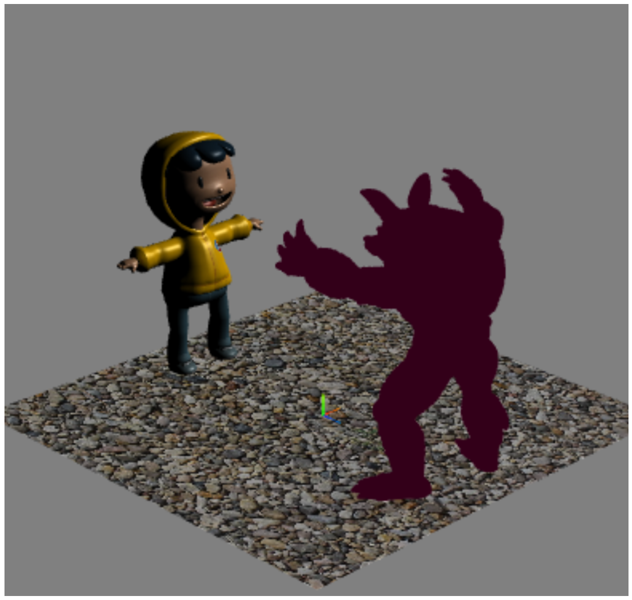

Figure 3: Question 1b). Texture mapping with ShaderMaterial.

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

A skybox is a simple way of creating backgrounds for your scene with textures. We have provided six textures in `images/pos[x/y/x].png & images/neg[x/y/z].png`. You will implement a skybox using cube environment mapping as discussed in class, where you map these textures on to a large cube surrounding the scene. You need to load the six textures to `skyboxCubemap` in the proper order, you can pass this as a `samplerCube` uniform to complete the `skybox` shaders. Sampling is done using the overloaded `texture()` function. To sample a `samplerCube` object, you require a texture and a direction. In this case the texture coordinate input is replaced by the viewing direction for the specific fragment.

For this question, you will be completing the `skybox.vs.glsl` and `skybox.fs.glsl`. Note that while the material and shaders are already loaded, it's your task to create the correct geometry and add it to the scene as you did in the previous assignments.

Hint 1: The skybox should move with the camera, so that it always is in front of the camera.
Hint 2: Cubemaps are sampled using a direction vector. Think about which vector you can use for this.

Figure 4: Question 1c). Skybox.

(d) **(20 points)** 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.

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?

Figure 5: Question 1d). Shiny armadillo.

(e) **(20 points)** Shadow Mapping.

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.). In this assignment, we will use shadow mapping. 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.

For this part, you are only required to use the functionality that is built into Three.js. However, you need to ensure that shadow camera is set up properly to minimize artifacts in the shadows (e.g., truncation, jaggies), especially as the light source is moving.

- You are not required to write a new shader for this question. Your task is to just setup a shadow camera. This shadow camera is essentially just the light source which is casting the shadows.
- Find the light source and set some attributes on it to enable it to cast shadows.
- Several pieces of the puzzle have already been filled in for you. For instance some models in the scene can already cast shadows. This is done by setting bool values on the respective objects.

Hint 1: You can use a Three.js `CameraHelper` to visualize the shadow map camera's view frustum.

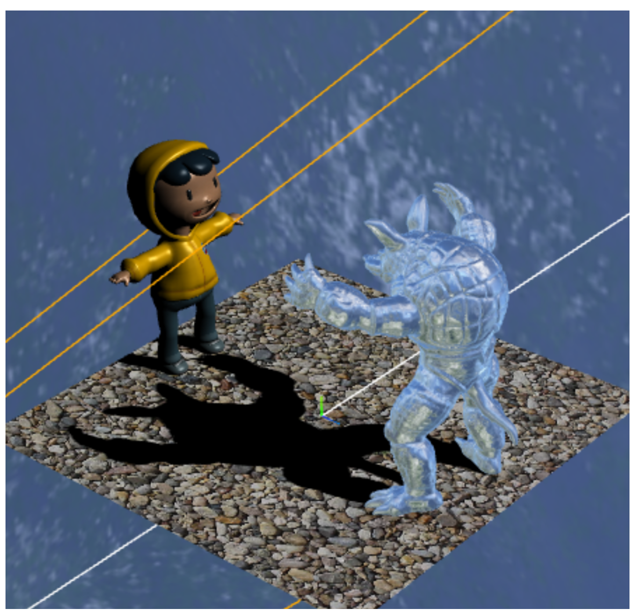

Figure 6: Question 1e). Shadow mapping.

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.

2.1 Hand-in Instructions

You must 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. Create a folder called "a4" inside your "cs-314" directory. Within this directory have two subdirectories named "part1," and "part2", and put all the source files and your `README.txt` file for each part in their respective folder. Do not use further sub-directories. The assignment should be handed in with the exact command:

`handin cs-314 a4`
on a department computer, which you can SSH into.
You may also use `Web-Handin` by following this link `https://my.cs.ubc.ca/docs/hand-in`, logging in with your CWL credentials, and writing "cs-314" for the course, "a4" for the assignment name, and zipping your assignment folder for submission.
It is always in your best interest to make sure your assignment was successfully handed in. To do this, you may either use the `Check submissions` button in Web-Handin, or using the -c flag on the command line `handin -c cs-314 a4`.