# CPSC 314
# Assignment 4: Textures and Shadows

Due 11:59PM, Mar 29, 2020

Figure 1: Completed Scene

## 1 Introduction

In this assignment, you will be learning all about textures. You will implement a skybox and a textured floor, as well as make your favourite character, Shay D. Pixel(our wizard), colorful

and cast shadows. The terrain will use basic texture mapping (for color), normal mapping (detailed bumps), and shadow mapping.

### 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/cpsc314-2019w-t2/a4-release.git
```

### 1.2 Template

In the assignment, you are provided with Shay D. Pixel on a plane and template shader files. We have defined lighting uniforms for you, as well as loaded texture images you will need.

## 2 Work to be done (100 pts)

1. **Basic Texture Mapping** (10 pts)

   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`, `images/ambient_occlusion.jpg`, and `images/normal.jpg` that you have to apply to the square "terrain" that Shay D. Pixel is 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 1`.

   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 while the ambient occlusion maps is used to model the visibility of the indirect lighting, represented by the ambient term.

2. **Texture Mapping with ShaderMaterial** (30 pts)

   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 `wizard.vs.glsl` and `wizard.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** : Here the texture is flipped on the y-axis, remember this while assigning your UV coordinates.*

3. **Skybox** (20 pts)

   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.

   ***TIP** : Follow the hints and comments related to Q3 in the template code.*

4. **Shiny Objects** (20 pts)

   Another interesting use of samplerCubes is **environment mapping**. This can be used to make the magic sphere (originally colored white) 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.

***TIP** : Follow the hints and comments related to Q4 in the template code.*

5. **Shadow mapping** (20 pts)

   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.) but we 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 (see also the example code "helloWorld3" shown in class). 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 can use a Three.js CameraHelper to visualize the shadow map camera's view frustum.

   - 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 the terrain (or the floor) is set to receive shadows, the models in the scene can cast shadows. All this is done by setting bool values on the respective objects.

## 2.1 Hand-in Instructions

You do not have to hand in any printed code. Create a README.txt file that includes your name, student number, and CWL username, and a summary of your work done. Create a folder called "A4" under your "cs-314" directory, and put all the source files, your makefile, and your README.txt file for each part in the respective folder. The assignment should run without any changes directly from your submission folders. The assignment can be handed in on a department computer, which you can SSH into, with the exact command:
`handin cs-314 A4`

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`.