Assignment 5: Texturing

In this assignment, we will explore how textures can elevate the realism of our renderings by capturing details on their surfaces. We are particularly interested in rendering very old objects to depict the trace of time. We will commence with a notably ancient subject—our planet, which is, indeed, 4.5 billion years old—along with our old friend, the Stanford Bunny. Let's begin!

1 Reading

In addition to getting familiar with GLSL programming and their built-in functions and data channels, please review texture and normal mapping course materials as listed below:

- Course Slides on Texture
- Tiger Book Chapter 11
- https://learnopengl.com/Getting-started/Textures
- https://learnopengl.com/Advanced-Lighting/Normal-Mapping

2 Starter Code

The starter code for this assignment, found under `assignments/a5/`, includes `main.cpp`, `a5_vert.vert` and `a5_frag.frag`. Upon successful compilation and execution, you should observe the silhouette of a bunny together with a sphere at the center of the window, both colored black, as depicted in Fig. 1. We highly recommend you read the provided starter code, primarily focusing on the fragment shader at `a5_frag.frag`. You may first review the code's structure thoroughly, paying special attention to the variables declared at the beginning and their respective types and mathematical meanings (in particular, we made a few changes regarding the lighting and shading setting, e.g., by setting the default point lights outside the `main()` function of the fragment shader). Also, before commencing your coding efforts, ensure you understand the seven steps outlined within the `main()` function.

3 Implementation Tasks

You will implement four texturing techniques, including checkerboard visualization, texture sampling, texture-light interaction, and normal mapping, as well as a customized creative expression task. Next, we describe the five implementation tasks with expected intermediate results.

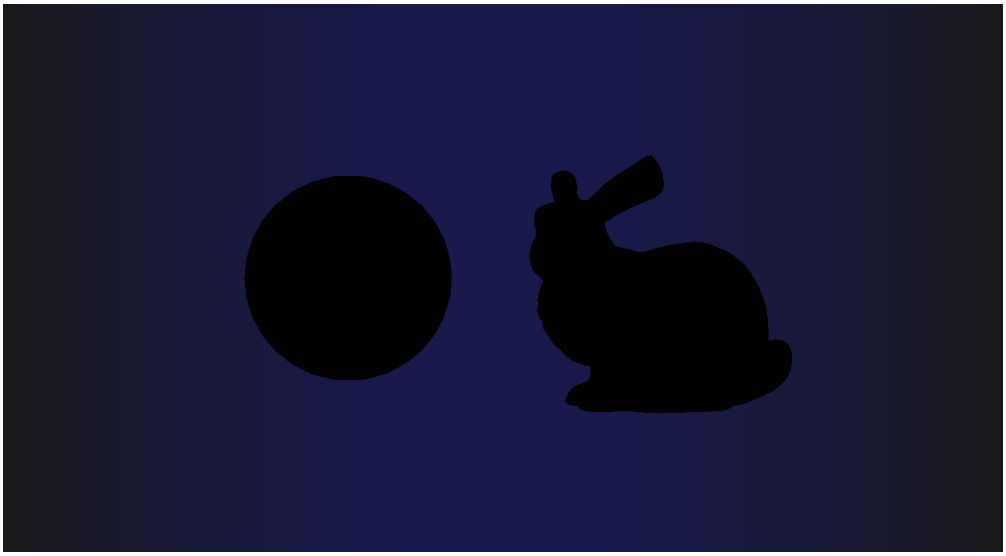

Figure 1: The default display of our scene setup with a sphere and a bunny.

Step 1: Visualize UV Coordinates with Checkerboard Pattern Implement a function `shading_texture_with_checkerboard()` that visualizes the UV mapping of a mesh using a checkerboard pattern. The function should use UV coordinates to calculate a color value that renders a checkerboard pattern on the mesh surface. As shown in Figure 2, we provide a reference result to help you check the correctness of your implementation for this step. We **do not require your result to match the figure rigorously**; as being said, you can freely choose your checkerboard grid cell size and the alternation of colors as long as your checkerboard pattern can reflect the local distortion of the UV coordinates on a mesh surface. In your implementation, you should consider using the `mod()` function in GLSL to allow grid cell color alternation.

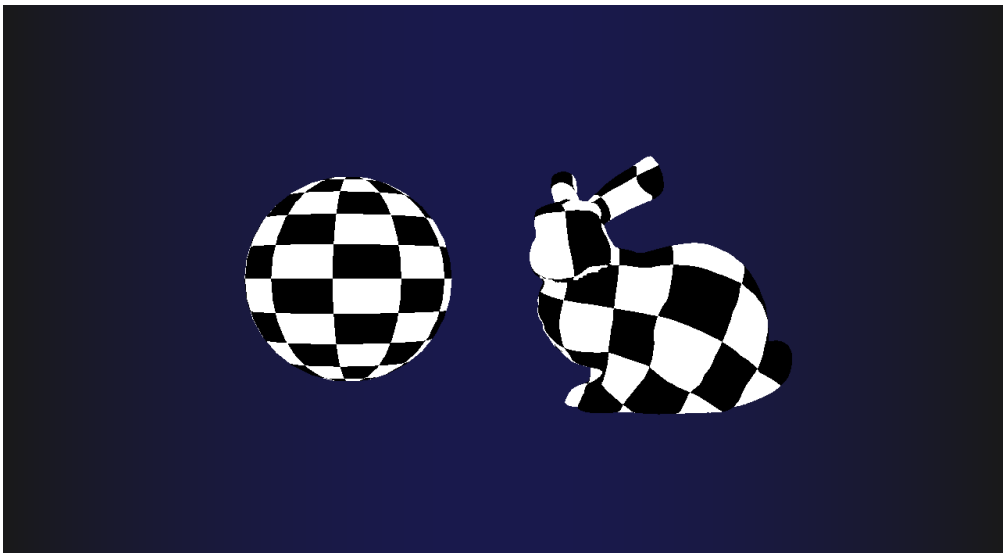

Figure 2: Visualizing UV texture mapping on mesh surfaces with a checkerboard pattern.

Step 2: Read Color from Texture Sampler Your next task is to develop the function `shading_texture_with_color()` to read texture values from the texture sampler using the UV coordinates and then assign this texture value to the output color. The texture is stored in the uniform texture sampler variable named `tex_color`. You should use the GLSL built-in function `texture()` to sample a color value from the pre-loaded texture sampler. If everything works correctly, you should see a texture-colored sphere and bunny, as shown in Figure 3.

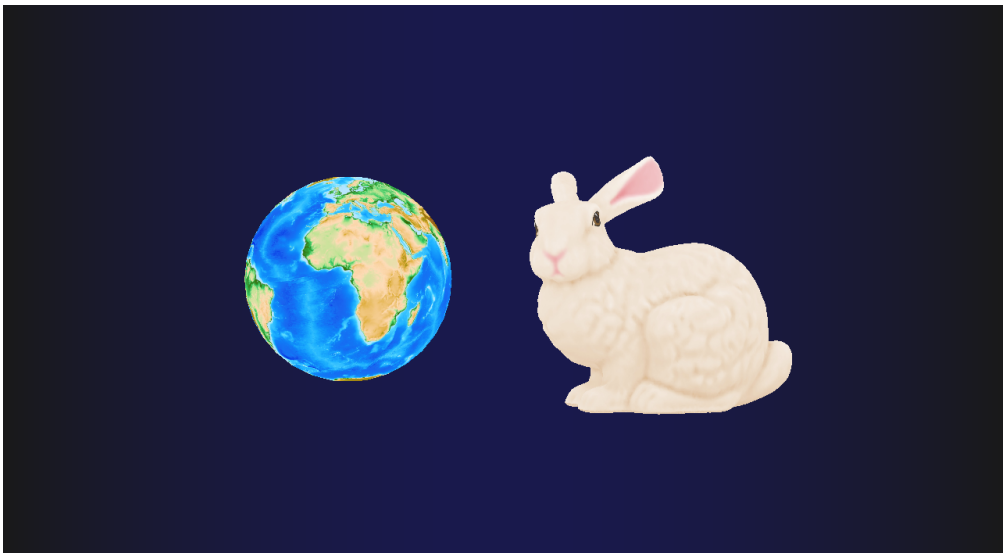

Figure 3: Mapping texture colors on the mesh surface directly with UV coordinates

Step 3: Phong Shading with Texture The next step is to synthesize the texture color with the shading model you have implemented previously. You will implement the function `shading_texture_with_phong()`. In this function, you are asked to implement a modified Phong model by multiplying the texture color into its diffusing term as follows:

$$\boldsymbol{L}_{Phong} = k_a \boldsymbol{I}_a + \boldsymbol{C} * (k_d \boldsymbol{I}_d \max(0, \boldsymbol{l} \cdot \boldsymbol{n})) + k_s \boldsymbol{I}_s \max(0, \boldsymbol{v} \cdot \boldsymbol{r})^p, \tag{1}$$

with k_s and $\boldsymbol{I}_s$ indicating the material specular property and light specular intensity (each as a `vec3`), $\boldsymbol{v}$ representing the direction from the object's surface point to the eye position, $\boldsymbol{r}$ representing the reflection direction from the light source, and p representing the power index controlling the shininess of the highlight region. Here we use $\boldsymbol{C}$ to specify the texture color read from the texture sampler, which is synthesized with the Phong shading color by simply multiplying its value with the diffusive term **component-wisely**. After your implementation, you should be able to get a textured Phong shading surface, as shown in Figure 4.

Step 4: Normal mapping Next, you are going to implement normal mapping in the same shader. This is the most challenging task in this assignment! Before starting your implementation, please read the course slides and supplementary reading materials carefully to ensure you understand the mathematical model of normal mapping.

The idea of normal mapping is to create a normal vector for each fragment based on the normal texture. Following this idea, you are asked to perform five sub-steps to correctly implement the

Figure 4: Phong shading with textured diffuse term

normal mapping algorithm. We briefly summarize the implementation details for each sub-step as below. You may find more information in the comment lines before each step in the starter code.

1. **Calculate the Bitangent Normal** The first step involves calculating the bitangent normal vector, derived as the cross product between the normal and tangent vectors. Namely, you need to calculate the vector $B = N \times T$ in your implementation. The bitangent vector **must be normalized** to maintain uniform vector lengths, ensuring accurate transformation of normals from tangent space to world space. Your implementation for this part will be within the function of `calc_bitangent()`.

2. **Calculate the TBN Matrix** Following the calculation of the bitangent vector, the next step is to assemble the TBN (Tangent, Bitangent, Normal) matrix. Each of the T, B, and N vectors constitute a column in this matrix, facilitating the transformation of texture normals to the world coordinate system. Your implementation for this part will be within the function of `calc_TBN_matrix()`.

3. **Read Normal Coordinates from the Normal Texture** This step is bifurcated into two tasks: (1) reading the normal vector from the normal texture using UV coordinates and (2) remapping each of its components from a [0,1] range to a [-1,1] range (As you might still remember, in our previous assignment we map a normal vector to a color, which is the inverse process of this step). Your implementation for this part will be within the function of `read_normal_texture()`.

4. **Calculate the Perturbed Normal with TBN Matrix and the Texture Normal** By multiplying with the TBN matrix, the normal vector read from the normal texture is used to produce a perturbed normal for each fragment in world space. This perturbed normal simulates detailed surface normals. Your implementation for this part will be within the function of `calc_perturbed_normal()`.

5. **Phong Shading with Normal Mapping** The last step is the integration of the perturbed normal into the Phong shading model. This involves putting together the functions you have

implemented to calculate the final fragment color. In particular, you need to call the implemented functions for calculating the bitangent vector, assembling the TBN matrix, reading and remapping the normal from the texture, and calculating the perturbed normal using the TBN matrix and the texture normal. The perturbed normal is then utilized in the Phong shading model to estimate the light and shade on the surface, considering the detailed texture provided by the normal map. Your implementation will be to call the previously implemented functions properly within `shading_texture_with_normal_mapping()` to calculate the perturbed normal vector for each fragment.

If everything happens correctly, you will see an old planet as well as a blobby Stanford bunny, both with tons of details on the mesh surfacees (as shown in Figure 5).

Figure 5: Phong shading with normal mapping

4 Creative Expression

In the Creative Expression section for this assignment, you will practice placing new mesh objects into the scene and setting textures for them. You want first to add your customized mesh objects and specify their transform and material (which you should be familiar with from your previous assignments) in the function of `Create_Old_Object_Scene()` in `main.cpp` by mimicking `Create_Bunny_Scene()`. You need to ensure your mesh objects have texture UV coordinates in their .obj files and check that these coordinates are read correctly (e.g., by checking the checkerboard on the mesh). Then you want to set the texture images for colors and/or normals by calling the functions of `Add_Texture_For_Mesh_Object()` (pay attention to how we used this function in the `Create_Bunny_Scene()` function). You want to adjust (or add) texture mappings within the fragment shader `a5_frag.frag` to create realistic geometric details on your objects. You are encouraged to combine the techniques you acquired from previous tasks to achieve a visually appealing rendering of old objects. Look for inspiration from previous works on Canvas).

The theme of this Creative Expression is **Trace of Time**. You are encouraged to convey the depth

of time through your work, emphasizing the profound impact of time by incorporating features such as erosion, scratches, and other markers of aging into your textured objects. You may try to evoke the historical essence and the wear of centuries, capturing the intricate details that reflect the passage of time in its most authentic form.

4.1 Texture Resources

Some links for downloading meshes with UV coordinates and textures:

- https://www.turbosquid.com/
- https://www.cgtrader.com/
- https://sketchfab.com/

As an example, if you choose to use TurboSquid to search for textured mesh, you can check whether the mesh contains UV mapping and texture file in the following place (see Figure 6) after selecting a mesh you are interested in. Other websites also have similar tags. If you just want some random textures/normal maps, an easy to try website will be polyhaven.

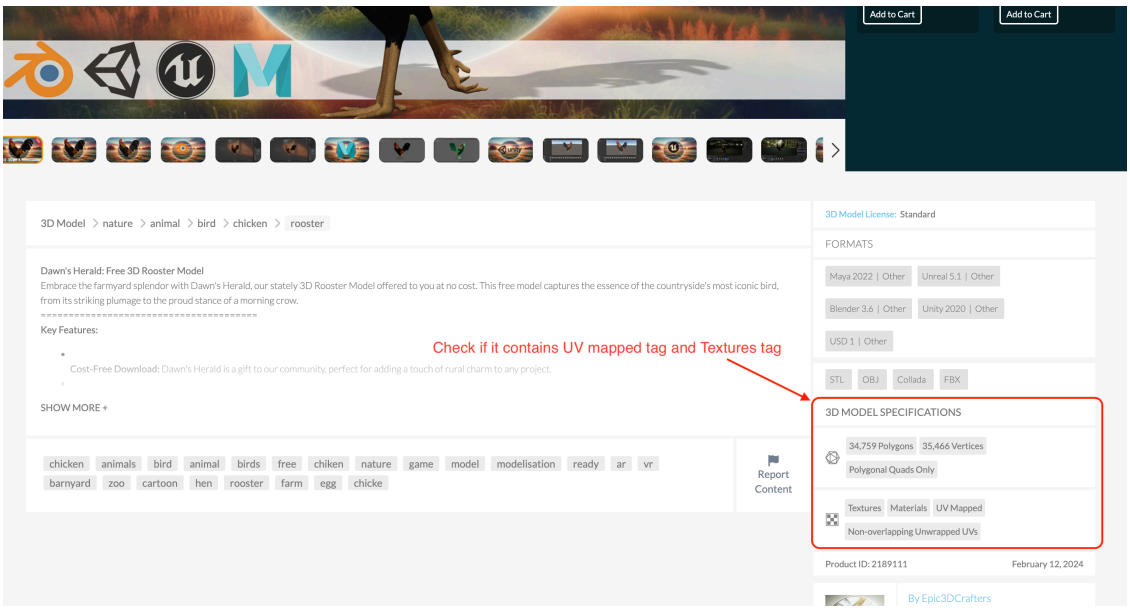

Figure 6: Check whether the mesh has a texture on TurboSquid

One caveat to notice is that the texture file you downloaded may be in format `.png`. **Make sure to convert it to `.jpg` before loading it to accommodate the implementation of our image reader. We also provide a set of textured meshes in the `./a5/textured_mesh` folder** and **a set of normal maps in the `./a5/normal_maps` folder**. See all textured mesh examples in Figure 7 and normal maps in Figure 8. Feel free to take them as you need.

Figure 7: Provided set of textured meshes

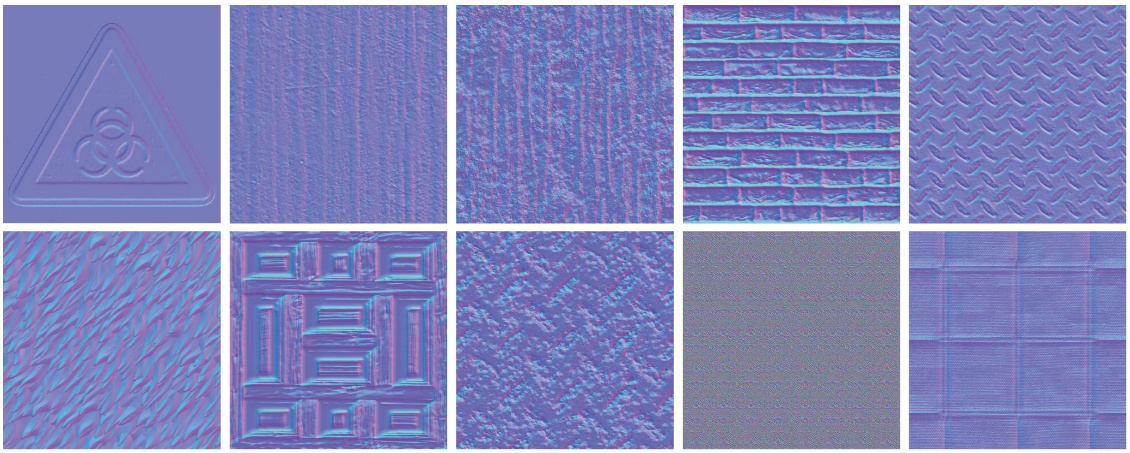

Figure 8: Provided set of normal maps

5 Submission

Submit the following components for evaluation:

- Your source code `main.cpp` and `a5_frag.frag`;

- Screenshots demonstrating the correctness of your implementation for Steps 1-4;
- A video or a few screenshots from different camera angles demonstrating your customized old object scene;
- A concise paragraph that provides a technical explanation of your implementation for the customized scene.

6 Grading

This assignment is worth a total of 8 points, with the grading criteria outlined as follows:

1. Technical contribution (7 pts): The core of the grading is based on the correct implementation of texture models. The distribution of points is as follows:
   - **Step 1 (checkerboard visualization):** 1 point
   - **Step 2 (texture color):** 1 points
   - **Step 3 (texture lighting):** 1 points
   - **Step 4 (normal mapping):** 3 points (0.5 point for each of the four sub-steps, 1 point for the final result)
2. Creative expression (**Step 5**) (1 point): This aspect focuses on your ability to create new textured objects by reading UV coordinates and applying them with different lighting effects.

7 Sharing your Work

You are encouraged to share your graphical work with the class. If you want to do so, please upload your image to the Ed post **A5 Gallery: Trace of Time**. This is an excellent opportunity to engage with your peers and gain recognition for your work. We look forward to seeing your demonstration of the trace of endless time!