GatorLand

Computer Graphics Final Project Report

Ben Rheault - 23511483, Sheela Ippili - 29950862

Initial Goals

The initial goals of this project were to experiment with more advanced graphics techniques than were used in earlier projects. We decided we wanted to implement texture mapping, normal mapping, and environment mapping. With these techniques, we hoped to be able to replicate an immersive virtual 3D environment that would be themed after the University of Florida.

We set out to find a somewhat low-polygon mesh of an alligator, and apply a texture of alligator skin to it. Given this mesh, our goal was to implement normal mapping on the texture to make the final result appear to have the topographical complexities of the hide of a real alligator. We also planned to implement environment mapping using a skybox that we would create by merging pictures we took somewhere on campus. This would be used to make this alligator appear to exist in a real environment, instead of an empty void.

What worked and what didn't

Given various restrictions related to time and logistics, we did not succeed in fully realizing our initial goals. To make up for this, we implemented other graphics techniques in place of those that did not work.

The first issue we ran into was with the skybox. Free software that can merge images into a skybox well is hard to find, and even then the pictures have to be taken fairly precisely. We did not have access to a quality camera or tripod, and trying to take pictures on any part of the campus meant people were always walking through the shot, so the skybox that resulted would have a lot of inconsistencies. In the end, we decided the focus of this project was on programming a skybox into OpenGL, not creating one of our own, so we found an available cubemap texture online to use.

The other, more problematic issue was with normal mapping. The first issue we encountered was that we could not find a decent alligator mesh online without paying for it. We also had difficulty finding a texture of alligator skin that came with a matching normal map, and the normal map we did find varied severely in pattern from the texture we had. Lacking the modeling skills to make such a mesh ourselves, and encountering other technical issues with implementing normal mapping, we decided to go without it in our project.

To make up for the loss of normal mapping, we decided to implement two other techniques: blending and instancing. Blending allowed us to include an alligator in our scene by using a two-dimensional image with a transparency channel, which let the gator blend into the scene. We improved on the effect of this scene by adding grass using the same blending technique. We used instancing to create many copies of this grass placed atop a ground plane with a dirt texture.

Finally, we added a dynamic camera. This allows a user to traverse the scene with the W, A, S, D keys and look around with the mouse. We attached the skybox to the camera's movement to prevent the user from traveling off-center or even outside of the box and breaking the illusion of the textured environment.

The Process

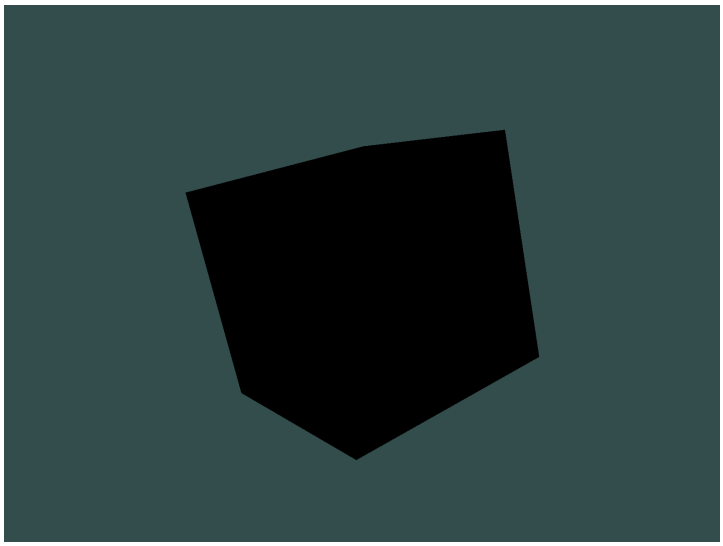
Many of the techniques used in this project were done with instruction from LearnOpenGL.com.

Texture Mapping

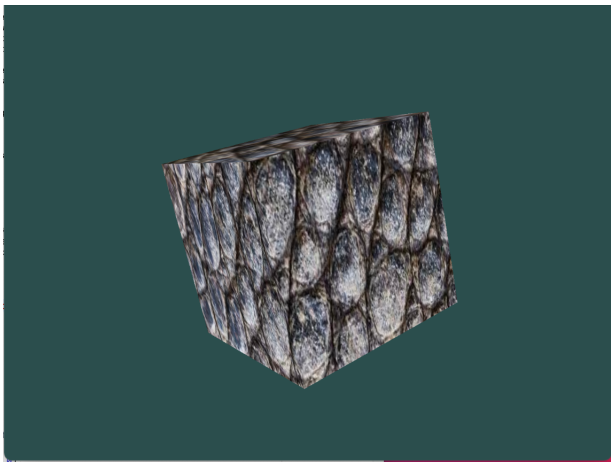
https://learnopengl.com/Getting-started/Textures

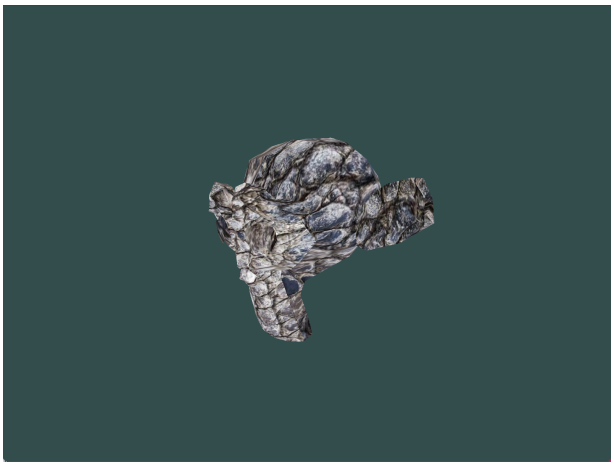
The ability to map textures to objects formed the base of this project, with all other aspects of it requiring the use of texture mapping in some way. Thus, we decided to make this the first aspect of the project we would complete.

Beginning with our base code from Project 3, we worked on applying texture to our pre-existing objects. Following the OpenGL tutorial, we installed a library called stb_image.h to be able to load any format of image file and apply it to a mesh. With our first few attempts, the texture would not load properly onto the mesh due to a failure to properly load the image we used. This led to the following image appearing when running our code:



After loading the proper image, we were able to place our alligator texture onto the cube, giving us this result with the cube and the monkey, respectively:





To accomplish this, the first thing we had to do was add UV coordinates to the OBJ file for the cube. That allowed us to load the texture and bind it to the faces of the object. We then used functions to generate a Mip Map of the texture to enable antialiasing for strangely-shaped pixel footprints. The UV coordinates of the texture are sent to the vertex shader, which sends them along to the fragment shader, which now retrieves the texture data instead of raw color data. The monkey OBJ file came with its own UV mapping, allowing us to apply the same

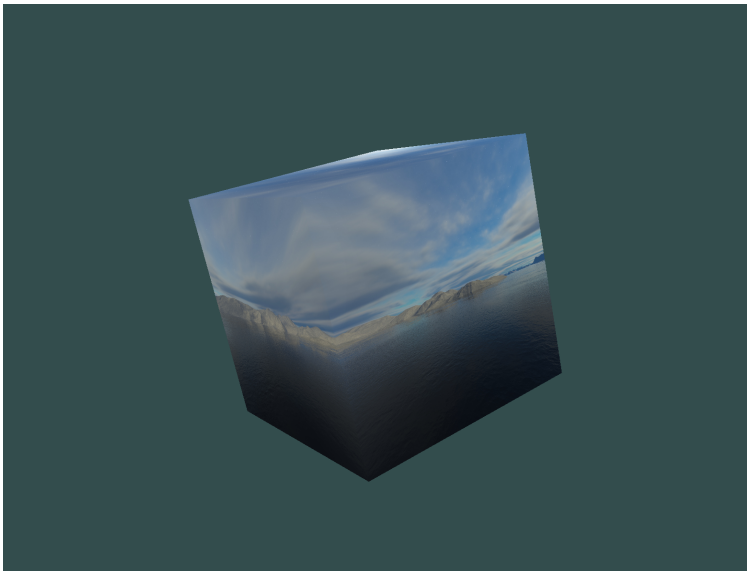
texture to a more complex object. Without a custom-made texture file for this UV mapping, however, it does not apply in a realistic way.
<Insert snippet of the texturing code>

Cube Mapping

https://learnopengl.com/Advanced-OpenGL/Cubemaps

To accomplish cube mapping, we had to be able to apply six different texture files to the six different sides of a cube. With the right textures arranged in the right way, this would allow us to create a realistic environment surrounding the camera.

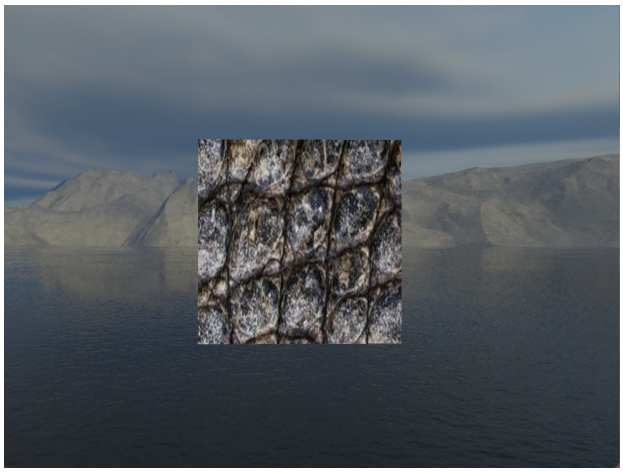
The code involved looping through a vector of texture paths to map each one to a different face of our cube. This also included the same mip mapping process from before. To implement this, we made two functions, one would load standard textures for our other objects, and one would load the skybox texture for the cubemap. This also required us to have two different vertex and fragment shaders, which we compiled individually. With this in place, we got the following result:



This worked, but it did not create the immersive environment we were looking for. For this, we needed to have the cube positioned perfectly centered on the camera to avoid any distortion from the viewing angle. Centering the cube on our camera position let us see the environment from multiple realistic angles, as shown here:



This left us with the trouble of getting two objects to show simultaneously in a scene. As it stood, we could load our textured object or the surrounding skybox, but not both. Solving this problem required us to create multiple Vertex Buffer Objects and Vertex Array Objects, and send them to their individual shaders to be processed. Once this was figured out, we were able to produce these images:



Environment Mapping

https://learnopengl.com/Advanced-OpenGL/Cubemaps

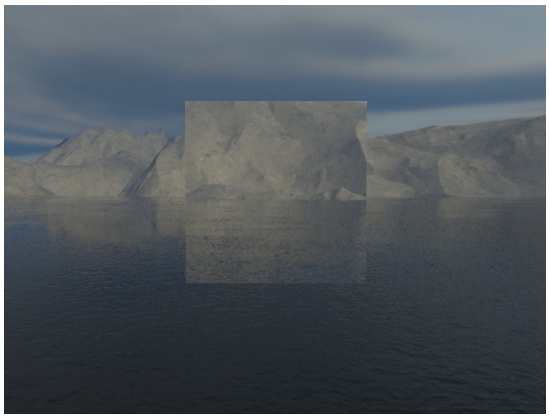
Environment mapping was a concept we were interested in as an extension on cube mapping. This would allow us to create objects in the scene that would reflect or refract their environment.

To create reflection, we used OpenGL to find a reflection vector given an object's normal vector. This we used as a vector to sample the cubemap, giving us the color to display on the object. Refraction is done similarly, again by creating a new view vector, just with a different OpenGL function to produce a vector that travels through the object, instead of bouncing off. This was done in the shaders using the following snippets of code:

```
float ratio = 1.00 / 1.52;
vec3 I = normalize(Position - cameraPos);
vec3 R = reflect(I, normalize(Normal));
FragColor = vec4(texture(skybox, R).rgb, 1.0);

float ratio = 1.00 / 1.52;
vec3 I = normalize(Position - cameraPos);
vec3 R = refract(I, normalize(Normal),ratio);
FragColor = vec4(texture(skybox, R).rgb, 1.0);
```

Implementing this functionality onto cubes in our scene produced the following results:



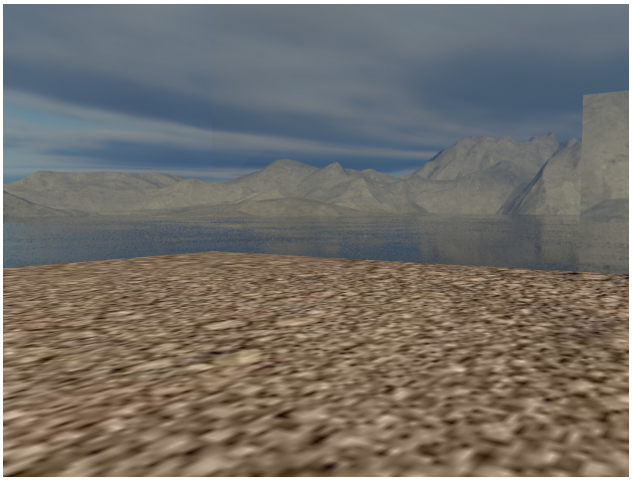
Blending

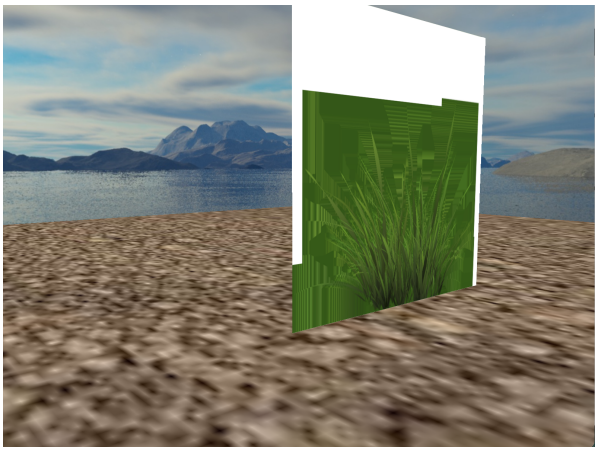
https://learnopengl.com/Advanced-OpenGL/Blending

Now that we had a realistic environment in which to create our scene, we wanted to start adding objects to it. Creating a floor plane was relatively easy. We edited a cube using Maya to delete all but one face, and used that OBJ file to create a surface below the camera. Applying a ground texture to this plane gave us a decent floor.

We could have extended the floor outside of the skybox to make it blend into the environment more, but extending the plane to this size would make our texture less recognizable as a dirt floor. This could have been circumvented by using many connected ground planes, each with an identical texture, but extending the plane outside of the skybox would reveal the cubic nature of the environment and contribute to breaking the illusion.

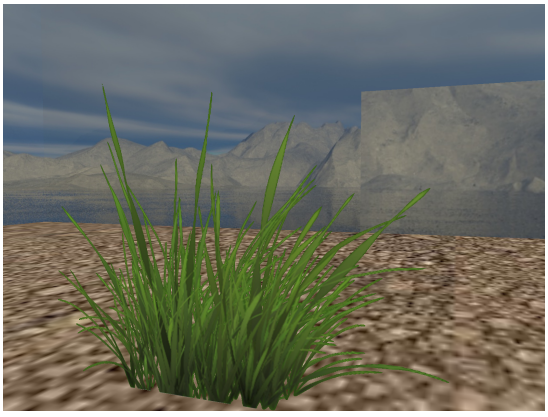
The next step was to add grass objects to the scene. Again, without access to quality meshes, we decided to compromise by applying grass textures to more planar objects, and place these objects throughout the scene. Our first attempt at this resulted in this:



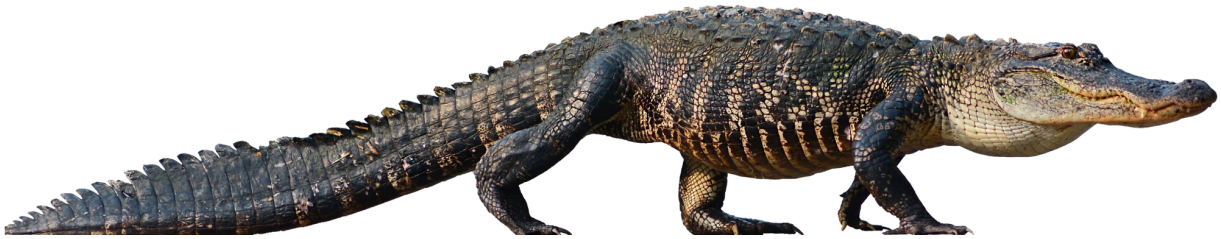


Obviously, this result was not acceptable and did not lend to a realistic environment. We came across blending as a solution to this problem. Blending requires an image with a transparency channel and allows us to make use of that channel to cut out the unimportant parts of the image.

This technique is implemented in the fragment shader, by checking if the average alpha value of the texels contributing to the fragment is under 0.1. If so, the fragment is deleted and does not show up on the screen. Finding an image of grass with an alpha channel was easy, and using this method we got much better results:



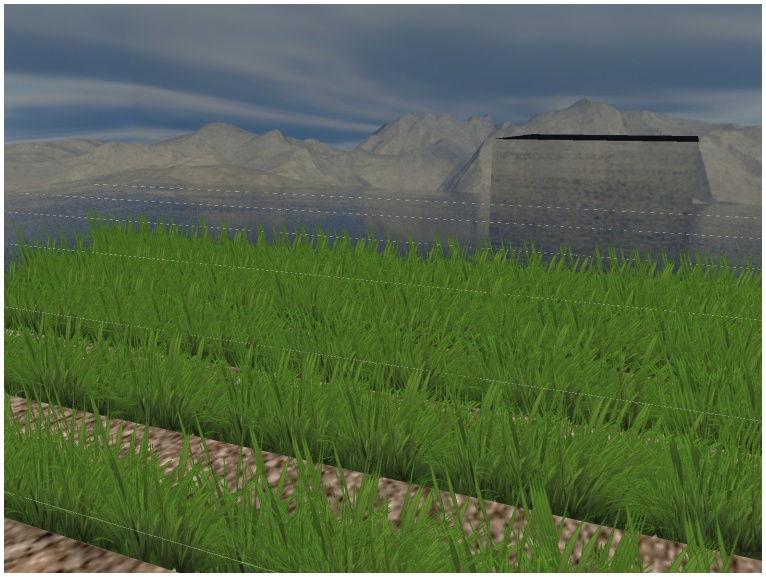
We decided to also implement the alligator using this method, but did not have a picture with the correct transparency. Instead, we used photoshop to create our own image, editing it to place the gator on the bottom of the texture to avoid it looking like it were floating above the ground plane.



Instancing

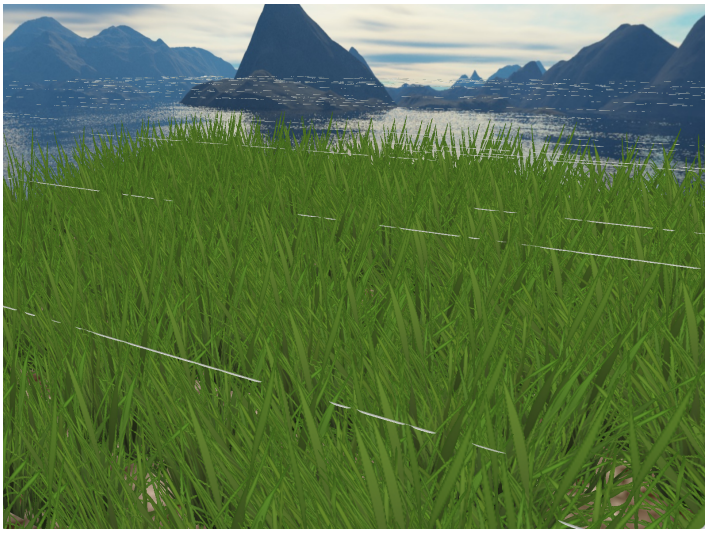
https://learnopengl.com/Advanced-OpenGL/Instancing

With only a single grass object in the scene, the gator would hardly look like it was hiding. To make a realistic scene, we needed to add many instances of this grass across the ground plane. To avoid loading the same model, again and again, we created copies of model matrices with random translations for the grass across the plane. By rendering the object many times with different matrices, we got this result at first:

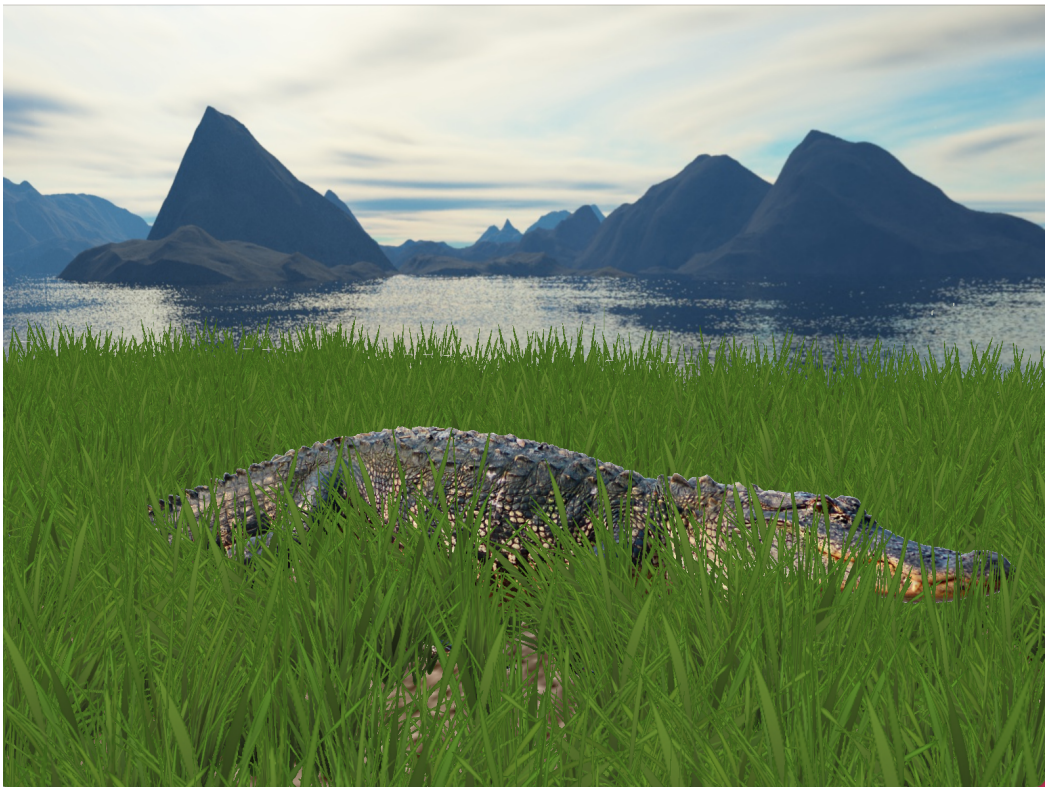


An error in our code caused all of the grass objects to be arrayed across only ten rows. We fixed the randomness of their placement with the following code, where the x and z values are both randomly determined to range across the ground plane, but the y value remains at 0 to make sure all the grass sits on the ground.

```
float x = (rand() % (int)(2 * offset * 100)) / 100.0f - offset;
float y = 0.0;
float z = (rand() % (int)(2 * offset * 100)) / 100.0f - offset;;
model = glm::translate(model, glm::vec3(x, y, z));
```



This gave us the following result:
While arrayed in this way, it is easy to tell that artifacts are being produced across the top of each object, a result of an imperfect texture image. We found a new Image to use to get rid of these, and placed the alligator in the scene to produce our final result.



Dynamic Movement

To get the full effect of this scene, the user must be able to traverse through it. We added dynamic camera movement incrementally throughout the development of the project. The end result is that the user can look around the scene freely by moving their mouse, and translate the camera through it using the W, A, S, D keys. We also decided to implement limitations on the user by preventing them from moving down through the floor plane. We created a minimum camera y-value of zero using the following code in the render loop:

```
if (glfwGetKey(window, GLFW_KEY_W) == GLFW_PRESS){
    if (cameraPos[1] <=0.0 && cameraFront[1] <=0)
        cameraPos[1] += 0.1;
    else
    cameraPos -= cameraSpeed * cameraFront;
}
if (glfwGetKey(window, GLFW_KEY_S) == GLFW_PRESS){
    if (cameraPos[1] <=0.0 && cameraFront[1] >= 0)
        cameraPos[1] += 0.1;
    else
        cameraPos += cameraSpeed * cameraFront;
}
```

We also made the skybox position dynamic. If the user is able to traverse to the edge or outside the skybox, they would destroy the illusion of the space. Having the skybox translate along with the camera keeps the correct perspective in place at all times.

One other attempt we made to make the scene dynamic was to have the grass rotate in place to follow the camera as it traversed the scene. This would prevent the user from being able to obviously tell that the grass is 2-dimensional. After working through the math that would get this to work, we attempted to implement it in a test scenario, but never got it functioning sufficiently well.

What we learned

This project allowed us to explore more advanced computer graphics techniques. Seeing the capabilities of texture mapping to add life to a scene was very interesting, and while we did not successfully implement normal mapping, we still got to explore the math and ideas behind it. Seeing how anti-aliasing and mip mapping are done also helped us to understand the topics covered in class.

Thoughts on Future Work

There are several ways we believe this project can be improved upon and expanded. The first is which would be to fill in the gaps left from our original goal. Being able to construct an environment map that more closely reflects the theme would be a great boost to this project. If

we were later able to make or obtain an alligator mesh, using normal mapping to make it appear realistic while cutting down on geometry use would be a useful and fulfilling exercise.

It would also be interesting to implement some kind of water texture to the scene, this would allow us to position the alligator half-submerged in the water, and show reflections of it on the surface. This was an idea we had in the planning stages of the project, but cut while writing the proposal.

We would also like to someday expand on the scope of this project. For now, the one floor plane we use only extends a limited distance from the initial camera position, not even far enough to extend outside of the skybox. Creating a larger plane with repeating ground texture and more objects to encounter would allow the user to explore more within this virtual environment.

Details of Contributions

Sheela was the lead project member for most of the programming in the project. She completed texturing on her own, and after that we met on many occasions to work together on the rest of the programming required for the project, including the creation of the skybox, blending, the translation and placement of all objects in the scene, and the continued attempts at normal mapping and instancing.

Benjamin was responsible for acquiring assets to be used for the project, and creating new ones if they were not available, such as the alligator image with an alpha channel. He also led the development of the dynamic scene elements such as camera movement and restrictions. He also attempted to implement the rotation of the grass objects to match the camera, to limited success.