Kevin Bajaj
EECS 672
12/11/14

Project 4 Report

Introduction

The goal of this project was to create a model of a 3D scene and render it. For my scene I decided to create a basic basketball court. This was chosen because I have spent large amounts of time watching and playing basketball throughout my life. In order to create this 3D scene I decided I needed the floor of the court, basketball goals, bleachers, benches, some surrounding asphalt, and a fence.

Model Design and Implementation

The first object that created was the court floor, which was rendered as a simple block. The function uses 9 parameters, the first three represent the lower left point of the court. The next three represent the desired length of the court in the x, y, and z directions and the last three represent the desired red, green, and blue values for the court.

The next object created was a basketball goal, to accomplish this I used the block function mentioned above as well as a subclass called disk that was used to render the rim of our basketball goal. The “Basketball_Goal” ModelViewWithLighting subclass takes four parameters, the first three are the lower left point of the basketball goal post and the last one is a boolean that can flip the backboard and rim to the other side in the x direction. This boolean was needed so that the second basketball goal would have the backboard and rim on the other side, as it would on a real court. The Basketball_Goal class then calls the block's render function to create the post and the backboard and calls the disk's render function to create the rim. These calls are all made using the default dimensions of the basketball goal, which are defined in the define_Basketball_Goal function.

The benches were the next thing to be added to the scene. The “Bench” ModelViewWithLighting subclass takes 6 parameters. The first three are the lower left corner of the top of the bench and the next three are the desired length of the bench in the x, y, and z directions. The Bench class then has default dimensions for the bench legs defined in define_Bench function. Finally, to create the bench three blocks are rendered in the positions specified by the parameters and the default dimensions.

The next thing added was the bleachers on either z direction of the court. The “Bleachers” ModelViewWithLighting subclass takes 7 parameters, the first three are the lower left point of the

bleachers. The next three, as seen in previous functions, are the desired length in the x, y, and z direction. The last one is a boolean used to flip the bleachers to face the other direction so they could be placed on both sides of the court in the z direction. The bleachers are created using four blocks stacked on top of each other with each one leaving a default “seat depth” in the z direction to create the seating area.

In this latest project I decided to add an asphalt surrounding area around the court and a fence around that surrounding area. These basic objects helped add to the realistic look of the scene especially since with the ability to use texture mapping.

Phong Lighting Model Design and Implementation

The Phong lighting model was implemented in the vertex shader and was designed to support multiple light sources. In this specific scene there are three light sources, the first is a weak directional light source designed to replicate the light from the sun near dusk. The other two are positional light source placed on either side of the basketball court, designed to replicate the common lights above outdoor basketball courts. The model differentiates between positional and directional lights by looking at the w value of the light source. If the w value is 0 then it is read in as a directional light source, if the value is 1 then it is read in as a positional source. The ability to add spot lights was also built into the phong model that was used in the vertex shader. The model recognizes a source as a spotlight if the spot cutoff is set to 90 degrees or below, this spot cutoff is default set to 180 degrees for other positional lights.

Dynamic Interactions

The ability to dynamically zoom in and zoom out using the mouse scroll wheel was added to the scene. This was done by implementing a handleScroll function in our Controller subclass, GLFWController. This function calls model view's addToGlobalZoom function with either a positive or negative value depending on if the user wants to zoom in or out. This addtoGlobalSZoom function then adds the increment value to the dynamic_zoomScale variable. Since this dynamic zoom variable is used to compute the ecXmin, ecXmax, ecYmin, and ecYmax values, it effectively zooms the view in or out on scroll.

The ability to dynamically rotate the scene was also added, this was done by adding a dynamic view matrix and multiplying it by the matrix created from the look at function. This new combined matrix serves as the mc_ec matrix to translate from model coordinates to eye coordinates. This dynamic view matrix is updated in the addToGlobalRotationDegrees function. The dynamic view

matrix is kept updated by using the cryph Matrix4x4 RotationDegree functions. There are 3 of these functions, one for the x value, one for the y value, and one for the z value.

Texture Mapping

Texture mapping allowed me to create much more realistic looking surfaces, the first object that was mapped was the court itself. For this is used a simple 1 to 1 mapping by setting the texture coordinates at (0,0), (0,1), (1,0), and (1,1) and binding them as per vertex attributes to the vertexes of the top face of that object. This was done at a 1 to 1 map because I wanted the court across the entire block.

The next objects I decided to texture map was the top of the benches in the scene. These were done by setting the texture coordinates to (0,0), (0, .4), (.4, 0), (.4,.4) and again binding them as per vertex attributes to the vertexes of the top face of the top of the bench. 0.4 was used here instead of a simple 1 to 1 mapping because the surface area of the top of the bench was so small that I wanted to shrink the texture down to make it look more realistic. In the end, the tops of the benches had 4 strips of wood across them and these appeared realistic to me.

Then I decided to place an asphalt texture on the surrounding area that I added. This was done in the same was as before except that I used a 5 to 1 mapping and set wrapS and wrapT to GL_MIRRORED_REPEAT to create the desired realistic effect.

Next, I texture mapped the fence surrounding the area. This was done using two separate classes since I wanted to texture both x sides on the side walls and both z sides on the north and south walls. To create the desired effect on the large walls a 5 to 1 mapping was used again with GL_MIRRORED_REPEAT. Since the walls near the entrance were smaller I used a 2 to 1 mapping for them, again with GL_MIRRORED_REPEAT. The last thing texture mapped was the face of the backboard that was facing the court. This was done in the same way as the court mentioned above.

Texture Mapping allowed me to create more realistic looking objects, so that is the main reason I took advantage of using it on many objects. However, the bleachers, rims, bench legs, and goalpost were all left without texture mapping.

Translucency

The main object I wanted to add simulated translucency to was the backboard, to give it the appearance of being glass. In my Controller Subclass the handleDisplay function was overwritten, but I was unable to adequately determine when it was drawing the translucent objects. I was trying to implement the translucency in fragment shader, but it was producing very unsightly results. Even after

writing a public boolean method that checked the per primitive variable “drawingTrans” I could not get it to behave correctly. To remedy this and still have the backboard have simulated translucency I moved many of the gl functions into my backboard class. Specifically, in at the beginning of the render function in the backboard class I call glDepthMask(GL_FALSE), glEnable(GL_BLEND), and glBlendFunc(GL_ONE, GL_ONE_MiNUS_SRC_ALPHA). I was then careful to disable GL_BLEND and set the depth mask back to true at the end of the function. This implementation allowed me to get the desired simulated translucent effect, however with one face also being texture mapped it still did not look as good as I would have liked.

Conclusion

Overall, I was able to create a realistic looking basketball court and surrounding area. Specifically, the addition of a few more objects and texture mapping in the latest project really helped the appearance. The benches were also moved to the surrounding area to give it a more realistic layout. One of the most difficult parts of this project was initially learning how the texture mapping worked, especially when binding an image to one or multiple faces of a block object. The other was correctly getting the overwritten handleDisplay function to know when an object was opaque or translucent. If this latter step could have been accomplished I believe I could have properly set the Mask and Blend functions in the handleDisplay function and then the fragment shader could have handled many different translucent objects without having to call these function in each specific class.