Jon Frodin  
Southern New Hampshire University  
CS-330-T4204  
4/16/2021

**Design Decisions**

*Navigation*

The user can navigate my scene by using the mouse to point the camera and by using some of the keys to move around. Scrolling on the mouse determines the speed at which the camera moves. The following keys have functionality built-in: A, S, D, W, Q, E, and P. In the same order, A moves the camera over to the left, S moves the camera back along the z-axis, D moves the camera to the right, W moves the camera forward on the z-axis, Q lifts the camera up along the y-axis, and E brings the camera back down along the y-axis. P switches between a perspective view and an orthographic view of the scene.

*Custom Functions*

I made a lot of custom functions for this project. I will attempt to go in order as they appear in the code. I won’t really address the shaders because I didn’t really create them or customize them so much as I duplicated them to add a second and third light. I did customize how the light displays, but not much else. The same goes for the textures. I added several (not counting the blank one used for black objects, I added 8) textures to make the scene livelier. I scaled down some of the images to improve loading time. The first actual functions that I added come around line 441 where I added the Q and E and P key functionality. I initially altered a header file for this functionality, but once I was informed that was not correct, I adjusted to do the same thing but without using the header file to abstract some of this.

The next custom function I made starts at line 596. This function is called getProjection and it uses a global variable to determine what projection to display. I did it this way because it was the simplest manner in which to switch from perspective to orthographic, even though I was trying to steer clear of too many global variables. Right after that function, I added renderLamp at line 606. This was because the render call within the main render loop was so huge and hard to keep track of. I wanted something that was optional because the effects of the light were there anyway without actually rendering the lamp. I wanted to use this when needed only. This cleaned up my render loop and got some of the unneeded code out of the main view.

Again, right after the previous function, I created my renderCube function at line 624. This one was altered week-over-week several times. Ultimately, as we did more things, I needed it to be more modular, so I took as parameters the rotation, translation, scale, and texture so that I could use one method for any cube. This function applies the lighting to each cube no matter where it is in the scene and will take the parameters and scale, rotate, and place a cube as directed. renderCyldiner on line 666 does the same, but for a cylinder. renderPyramid also does the same, but for a pyramid, on line 709. renderPlane does the exact same thing as renderCube, however since I was using different texture coordinates than renderCube, I wanted to use a separate function (line 753). I would love to learn how to do that from within a render function, but I simply lacked the requisite time. That brings us to my first heavily-customized funciotns: renderWater1 and renderWater2 on lines 840 and 797, respectively.

renderWater1 takes a clear image and draws it on the front of a cube using the same sort of functionality as the renderCube function with one crucial difference, renderWater1 turns off the depth mask first, renders the face of the cube, and then turns the depth mask back on. This makes sure that the front does not become just a black plane rather than a clear image. Turning off the depth mask was critical. Then, renderWater1 calls renderWater2 and uses a different texture to draw the rest of the cube.

Next up we have, in my opinion, my most heavily customized functions. I wanted to bring my scene to life so I made a struct called “Fish” on line 887 and created 3 instances of Fish on lines 893-895. The properties are all relevant to what comes next. Line 897 has the function getFishRotation which takes a Fish as a parameter. This determines, using the size and startStatus of a given fish, how to rotate that fish and then outputs that rotation to be used by the calling function. The critical aspect of this function, though, is turning the fish and updating the startStatus when necessary. So, if a small fish reaches or passes -1.0 on the x-axis, then this function rotates the fish in the opposite direction and changes the status to be used both by this function and by the positioning function later. The same goes for the other side. This is what keeps these moving fish within the bounds of the tank.

This brings us to getFishPos on line 941. Similar to getFishRotation, this takes a Fish as a parameter and makes adjustments to the incoming fish and then outputs the position this time, rather than the rotation as in the previous. This one uses some conditionals that I would love to revise, but they function as needed. They take the current position of the fish and update that position both in the source (the global variable submitted each frame) and the output to the screen. This brings us to renderFish on line 980. renderFish works like all of my other render calls with the exception of the parameters and the static rotation/translation. The scale, though, is static because I did not want the fishes getting bigger and smaller as they swim around. This is the parent function that calls the getFishPos and getFishRotation.

Finally, URender was pretty heavily customized to account for adding the static translations, rotations, and scales of my objects and then calling the various render functions needed to render the scene as a whole.

*Justification*

Some of the justification is very straightforward. For example, the requirements stated that Q and E should, when pressed, move the camera up and down. The requirements did not specify which should do which, so I just coded them as I saw fit. The same goes for the mouse scrolling. Some of the decisions are less straightforward. For example, I used cubes extensively to create my stand and its sub-parts. I chose to do this to simplify my vertices (which already contain a ton). I also did this because I think it is more modular and efficient to re-use the same cube however many times than to re-create a dozen or so cubes from scratch. I took all of the parameters I needed in my render function, though after thinking about it I believe I could have left rotation alone and only passed translation and scale where needed. If I had done this, I would have only had to modify which part was stretched and in what manner. That may have been more efficient than declaring rotation so many times.

The decision to add a driftwood pyramid was also directly influenced by the requirements. I could have made it any different shape, but I needed a pyramid to satisfy the requirement of using at least 4 basic shapes. I am not sure that was entirely necessary since I did use the complex shapes of my fish, but I wanted to fulfill everything listed. The decision to make fish and to also make them move was a costly one. That took a significant amount of time. I justify that by highlighting the fact that we shouldn’t be doing the bare minimum and I really wanted to demonstrate an understanding of the render, frame-over-frame. I also really wanted to bring the scene to life. Putting fish in there, even though I did not need the shapes anymore and definitely did not need the movement, seemed a perfect way to wrap this project up.

I chose the objects I did because they fit. I spent a good amount of time thinking about what shape would be best for each object. The fish shape was custom, and if I had more time to make it more realistic and perfect, I definitely would have. I would also have loved to add a turtle. In the end, I am very happy with the scene I developed and I think even the smaller, less-apparent details like adding another cube with a pebble texture ended up being a smart call overall. Going forward, I would really like to learn more about normals, textures, and application of those elements along with the lighting. The examples given in the course materials were so detailed. I did not really come close to that level of quality, but I gave this my honest best shot. In the end, I am very happy with the finished product despite many imperfections.