Paul Kline
EECS 700 Graphics
Project 3 Documentation
"Solomon's Temple"
11/19/2014

The idea behind your project and/or where you got the idea.

The idea behind my project remains unchanged. However, I did add grass to my scene. I'm not sure where I first got the idea to add grass, I just thought it would be really cool for whatever scene I do to have grass in it. Implementationally, I got the idea to use a Bezier curve from a Stumble Upon page that showed Bezier curve construction animations.

How you generated your model.

- I have corrected the constructors to not call the destructors of contained objects, but rather call delete on them and let the underlying calls call the destructor.
- I changed some of the geometry so that the cones along the front and back of the temple are more reasonably spaced. i.e. they now appear to have the same spacing as the cones going down the sides of the temple (all this being on the top). While they *appear* to have the same spacing, it is actually just close. This is necessary to fit an appropriate number of cones.
- (*Window changed again to be public member of GLFWController*)
- I have added the options to change the view amongst orthogonal, oblique, and perspective. This was done by subclassing the GLFW controller, implementing the method that handles key presses, , and changing the ModelView static varibale of projType. Then projType is used in the getMatrices method to construct the proper ec_lds matrix.
- Dynamic zooming was done in a similar fashion whereby I implemented listener methods in the controller class. This threw me for a loop for a while, but eventually I got it working. My first attempt at dynamic zooming was to multiply the ec_lds matrix by a scaled identity matrix. This performed the desired zooming effect, but the scene experienced clipping. Then I remembered/ was reminded by a fellow student that dividing the ec max/mins by a factor was the way to go. After this, the clipping problem with zooming went away.
- I then worked on dynamic rotation. While I had dynamic rotation in my last assignment, the way it was implemented was fundamentally different-- i.e. moving the actual mc coordinate of the eye. Dynamic rotation I found tricky. And by tricky I think I mean touchy and pick. For instance, rotating the scene up and down once the scene is 'sideways' seemed to have behaviour dependent on which order I multiply the x and y rotation matrices--
  i.e. xy != yx . Also the way in which the mouse location was tracked seemed to produce different effects when intuitively I thought the effect should be the same. Method 1 was me changing the dynamic view matrix by the delta x and y rotation matrices where the delta x and y where from the last time this method was called. . The second method, the one that worked, kept track of the overall (instead of the incremental) delta x and y-- the delta x and y from when the left

mouse button was clicked. This seemed to produce much more intuitive movement.

- The Phong lighting model somehow turned out to not be as bad as I was expecting. I simply went through my notes for the lighting model, and the translation to code was not too bad. I arbitrarily chose to make the lighting model in the vertex shader instead of the fragment shader. Then I created 3 light sources: one a nice directional blue, and the other 2 are strong red locational sources inside the fire pits. I should also note that for correctness of the firepit light sources, lights are defined in model coordinates and converted in the vertex shader.
- Once I got to this point of meeting project specification, I decided to work more on the grass class I had started because that would be cool. A blade of grass is defined by an up vector which specifies the main direction, an out vector which specifies the direction the glass blade curves towards, lengths for the previous two vectors, and a color. Additionally, grass exhibits attenuation in the sense that typical grass comes to a point. In my grass class, width attenuation can be specified by a number of functions including constant (for a perhaps “mowed” effect where the top is flat), linear (triangular grass), quadratic (a good normal grass effect), and cubic (I think the best and most realistic width attenuation). Also the blade of grade can be given a number of radian to rotate as it ‘grows’. This radial twist is similar to width attenuation functionality in the sense that the rate at which the overall rotation of the blade is applied can be any of the previously mentioned width attenuation functions. This provides some pretty cool twisting effects. Then I randomly placed about 8500 blades of grass with (appropriately) randomized parameters on my ‘ground’ block. Each grass blade is also a random shade of the color “forrest green.” All this in combination provides a realistic looking lawn.

The way you met project specifications.

1.
    a. New project 3 directory included and ModelViewWithLighting placed in it. mvutil replaced with project 3 version.
2. Interactive viewing:
    a. getMatrices does not do unnecessary calculations.
    b. diagonal of scene is calculated and used as “bounds”
    c. Dynamic rotation and zooming is implemented in an intuitive fashion.
    d. The view can be changed between oblique, orthogonal, and perspective respectively by hitting the ‘Q’, ‘O’, and ‘P’. Note that they must capitals.
3. Phong
    a. Phong lighting model implemented with 3 light sources: 1 directional blue, and 2 local red sources.
4. Additional Geometry: grass

Paul Kline
EECS 700 Graphics
Project 3 Documentation
“Solomon’s Temple”
11/19/2014

All the interactive controls your program supports so that I know how to use them.

- CAPITAL ‘Q’, ‘O’, and ‘P’ change the view to Oblique, Orthogonal, and Perspective respectively. Again note that they must be capital. This also makes it so it does not interfere with the code from project 2 where ‘o’ moves the eye out, ‘i’ moves in, and ‘a’, ‘s’, ‘d’, ‘w’, move the eye position left, down, right, and up respectfully. This actually changes the position of the eye which can give nice views, but makes for pretty weird/non-intuitive movement when used in combination when the dynamic_view matrix.

The thing (or things) that caused you the most difficulty while developing the project.

Most of it. The zooming wasn’t bad, but getting the phong lighting model (cpu side) proved challenging. For instance, I always got an “invalid operation” GLFW error whenever I tried to send the lighting model information to the GPU from a “super renderer” (one that just calls other render methods). But sending the lighting model from primitive renderers worked just fine.

Dynamic viewing also proved difficult to fish out the correct answer. I think for a lot of this graphics stuff, the conceptuals make sense, but there are just so many tiny implementation details that it’s easy to get lost.

Also, all the sudden, without changes, my program stopped being runnable. It would compile just fine, and then would close itself immediately upon running it. I then restarted the computer I was on, only to not be able to log into it anymore. I changed computers and recompile the program. It still closed instantly upon running it. I then commented everything out of the main method. Then it ran and didn’t close itself immediately. Then section by section, I uncommented the code and ran it. And it worked each time. I did this entire the project was back in it’s original state that made it run and crash, but this time, it did not run and crash. This seems like undefined behavior to me. My one theory is that ctrl-c ing the program instead of X ing it leaves the GPU in a bad state and after however so many ctrl-c’s it wouldn’t even run a program. But this does not explain why it exemplified the exact same run-crashing behaviour on the second computer.

Paul Kline
EECS 700 Graphics
Project 3 Documentation
“Solomon’s Temple”
11/19/2014

Any unique things you did in the project that you especially want me to notice while grading.

The grass! please see the first section about How I generated my Model to see how cool the grass is. Below I enlarged a grass blade for easier viewing and the lower image is a bunch of grass.