Lab 03 Lights & FX

**Fufillment**

* I have met all the requirements for the lab, and even did some extra this time involving mouse input

**Execution**

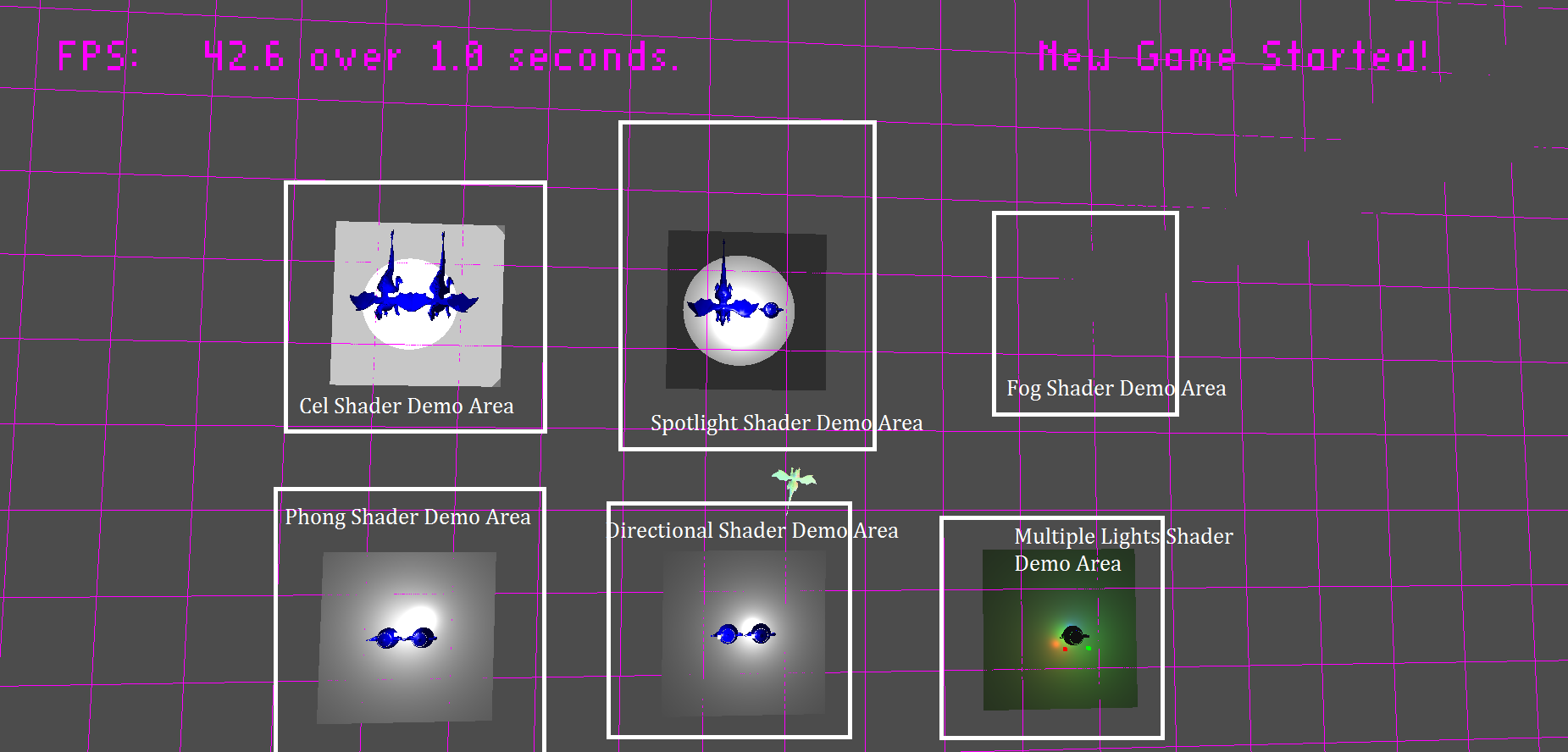
* I do not believe there should be any unexpected requirements for running

**Controls**

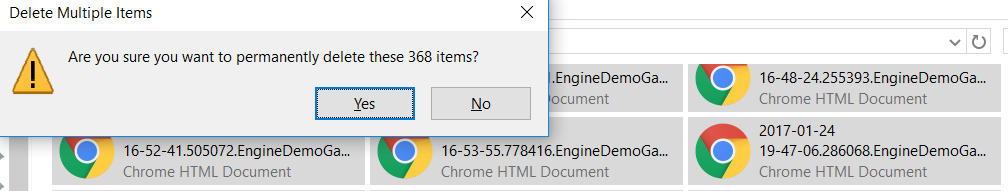
* Pressing the X key will close the application
* Pressing the P key will pause the application, pressing it again will un-pause the application
* Pressing numpad 0 will re-read the config file
* Pressing M, L, T or C will dump engine info to the console, this is pretty much exclusively used for debugging
* Pressing W will rotate BetterDargon to the left, S will rotate him to the right
* Pressing A will tilt BetterDargon forward, D backward
* Pressing Q will roll BetterDargon to the left, E to the right
* Holding space will move BetterDargon forward, in the direction he is facing, releasing will halt movement
* Right clicking and dragging the mouse will turn the camera around BetterDargon
* Scrolling in or out with the mouse wheel should zoom the camera accordingly, up to a minimum or maximum distance
* Pressing the 0-7 will set the light movement states as described in the requirements for the lab
* Holding shift will speed up the spot light and cell shader light change rates when the key is pressed
* Page down decreases the number of levels used in the cell shader by one, by ten if shift is held
* Page up increases the number of levels used in the cell shader by one, by ten if shift is held
* Pressing [ will decrease the cutoff angle of the spotlight by one degree, ten if shift is held
* Pressing ] will increase the cutoff angle of the spotlight by one degree, ten if shift is held
* Pressing \ will cycle through the attenuations in the config file, looping when the end is reached

**Screenshots (NOTE: Humorous comments included deliberately in addition to normal descriptions to make this doc more enjoyable to read. ☺)**

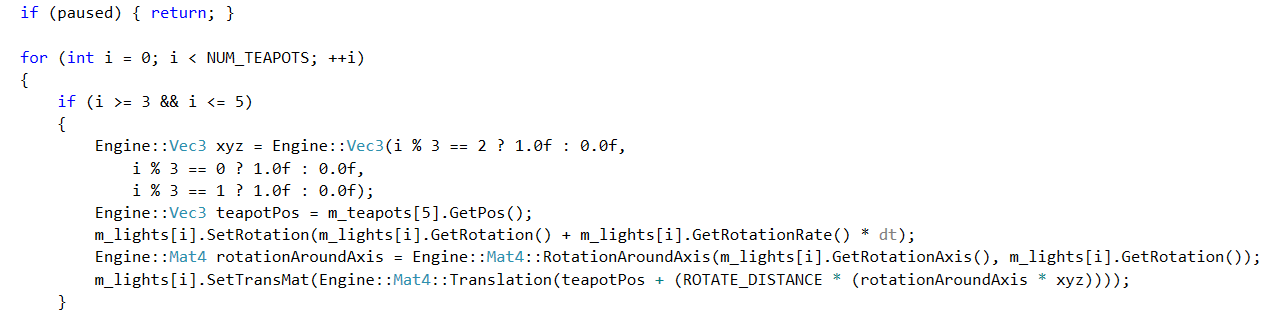
* **MISCELLANEOUS SCREENSHOTS**
  + The first screenshot shows yet another awesome overview of the universe in which this lab takes place. To the left, we can see the cell shader demo area and the phong shader demo area. In the middle lie the spotlight demo area and the directional demo areas. Finally, to the right, we have the fog shader demo area and the multiple lights demo area. Note that the objects in the fog shader demo area cannot be seen because we are far enough from them that the maximum fog distance has been exceeded.

****

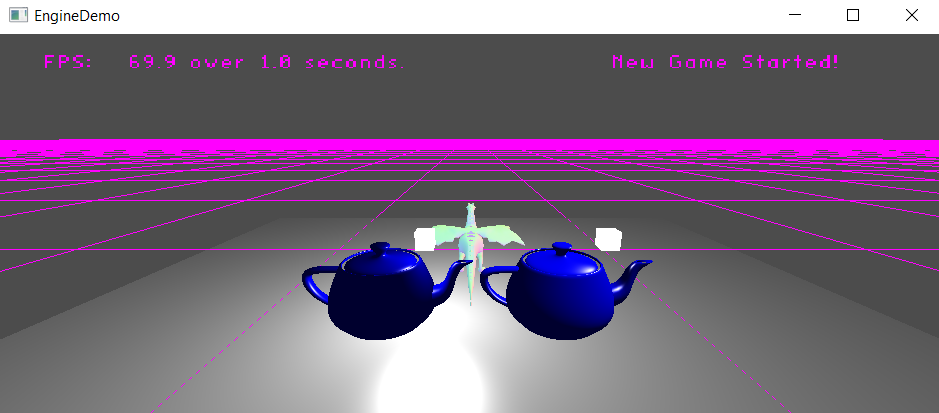
* + The second miscellaneous screenshot shows me clearing out my log files after doing this lab – maybe it’s useful for gauging how much effort/time/bugs/testing went into the lab somehow. Either way, that’s a lot of logs!



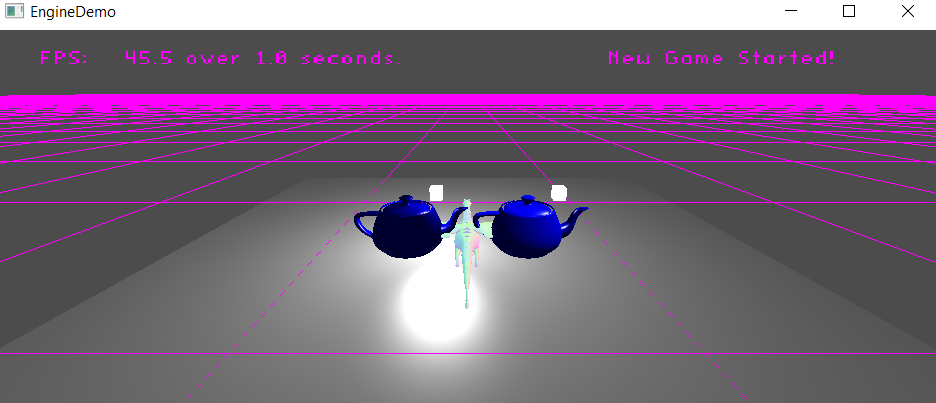
* **PAUSE SCREENSHOTS**
* The first screenshot shows pause code! I know, it’s super complicated, so I’ll explain it simply: Don’t rotate the lights or move them based on mouse input while the game is paused. Located above the code snippet shown is code that handles things that should happen regardless of the paused state of the game, like movement for viewing the world and updating the pause key itself so that the game can be un-paused. Maybe that’s important, maybe not. ☺



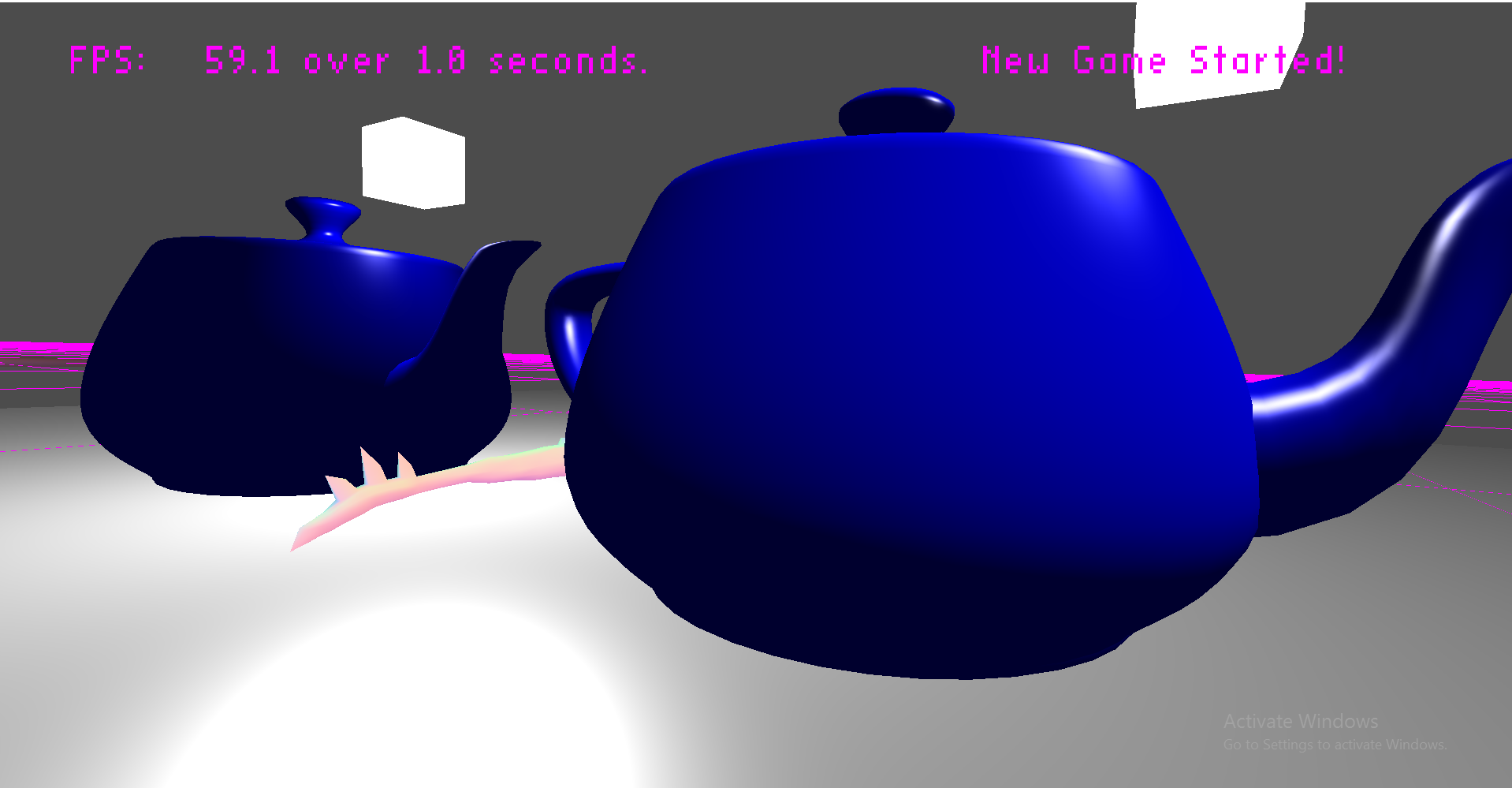
* To further demonstrate the functioning pause feature, I have a screenshot of the game running while paused – notice the fps is higher than while not paused (70 > 45) (side note: the fps is lower than in class because at the time of taking this screenshot I was not connected to a power source)



* Here, for reference, is a screenshot of the game running while not paused. Again notice the fps difference (45 < 70). This difference indicates that there is less processing going on while the game is paused, as would be expected of a working pause feature.



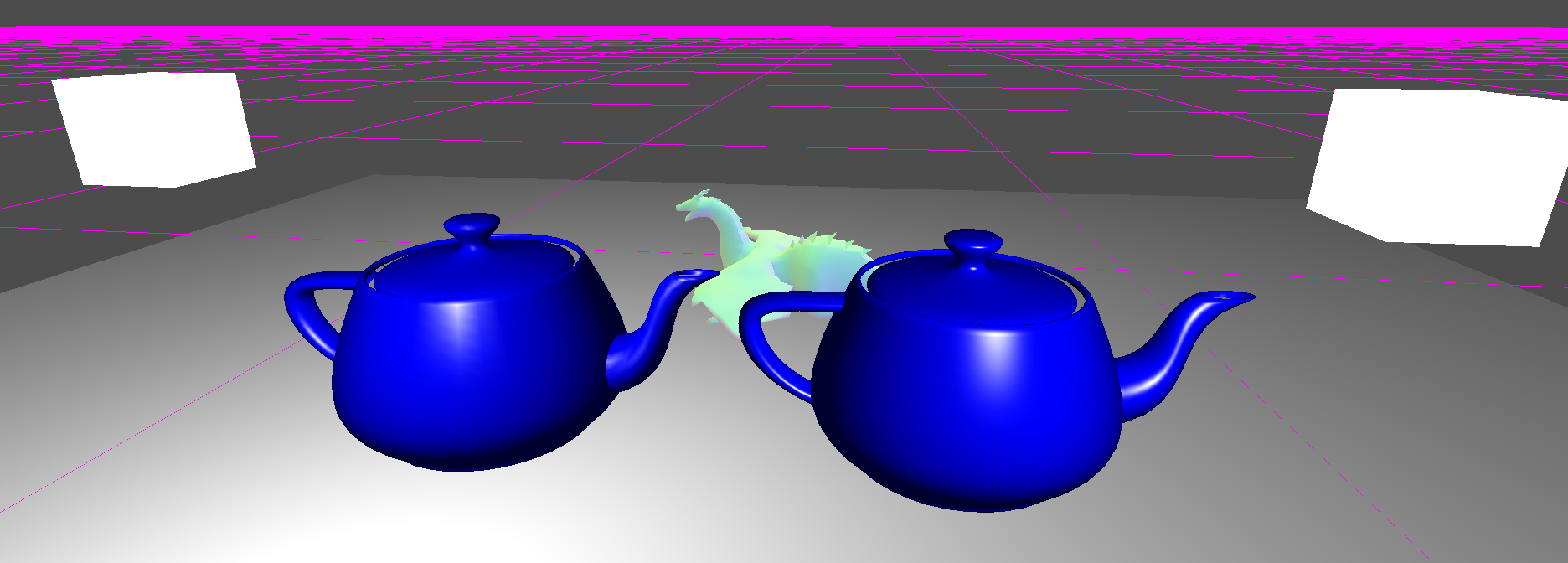
* **DIRECTIONAL LIGHT SCREENSHOTS**
  + In the first directional light demonstration screenshot, the difference between the point light (effects visible on the teapot on the left) and the directional light (effects visible on the teapot to the right) are made quite obvious due to the close up on teapots whose lights are positioned very close to the teapots themselves. Notice the teapot to the left has more area lit only by ambient light, because the triangles are lit based on the position of the point light, whereas the teapot on the right has less triangles only lit by ambient light. This is because the triangles are lit based on the direction of the light as opposed to its position.
  + Also feel free to note the deliberate maneuvering of the camera to a position intended to optimally show how the differences – not only is it closer, but it is angled such that both lights and both teapots are visible and taking up as much screenshot real-estate as possible. Also, I think the blue color of the teapots conveys an undertone of sadness and depression essential to understanding the piece of art as a whole, or maybe the developer just felt like blue was the right color for this. Guess we’ll never know.



* + Ah, yet another code screenshot. This shows that the fourth component of the teapot’s position is being set to zero to indicate that it is affected by directional light. The other teapot in this demo does not have this value set and is thusly using positional lighting as it defaults to one. This is indication of meeting that requirement for directional lighting.



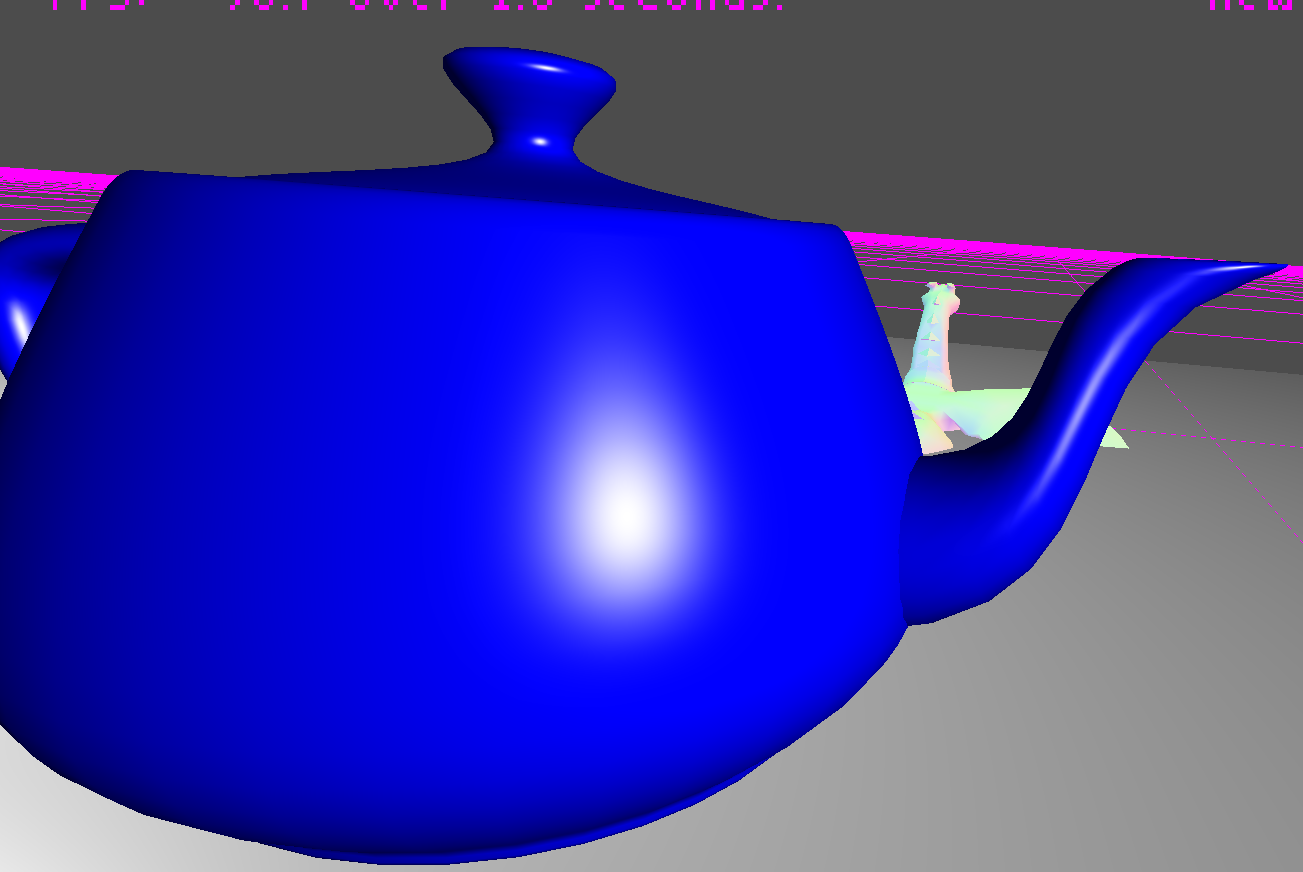
* **PHONG SHADING SCREENSHOTS**
  + The first demo screenshot for the Phong section shows a Gouraurd teapot (to the left) and a Phong teapot (to the right). The difference between the two shaders being whether lighting is calculated on a per-vertex or per-fragment basis, there is little difference except for with the specular highlight, which is much smoother on the Phong teapot (right).



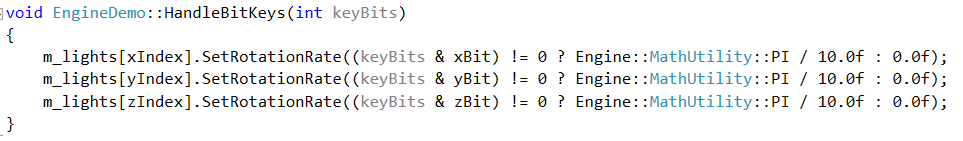
* + The next screenshot shows an extreme close-up of the Gouraurd teapot, to emphasize the horrid specular highlights that would look so much better if they were only done in the frag shader…



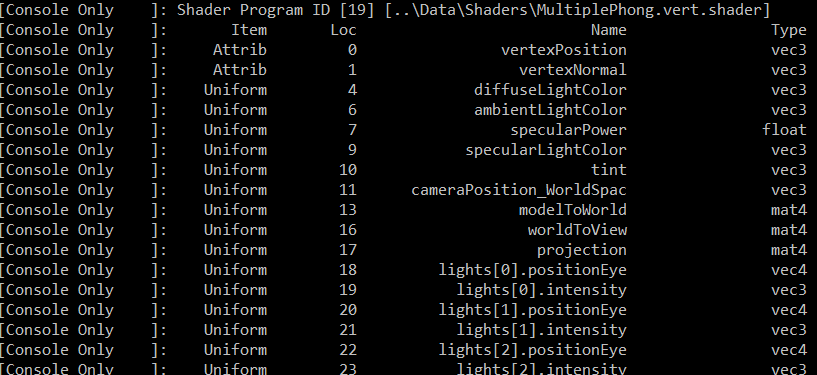
* + The final Phong Demo Screenshot shows an extreme close-up of the Phong teapot, with nice, cool, smooth specular highlights computed on a per-fragment basis. Doesn’t it look so much better than Gouraud shading?



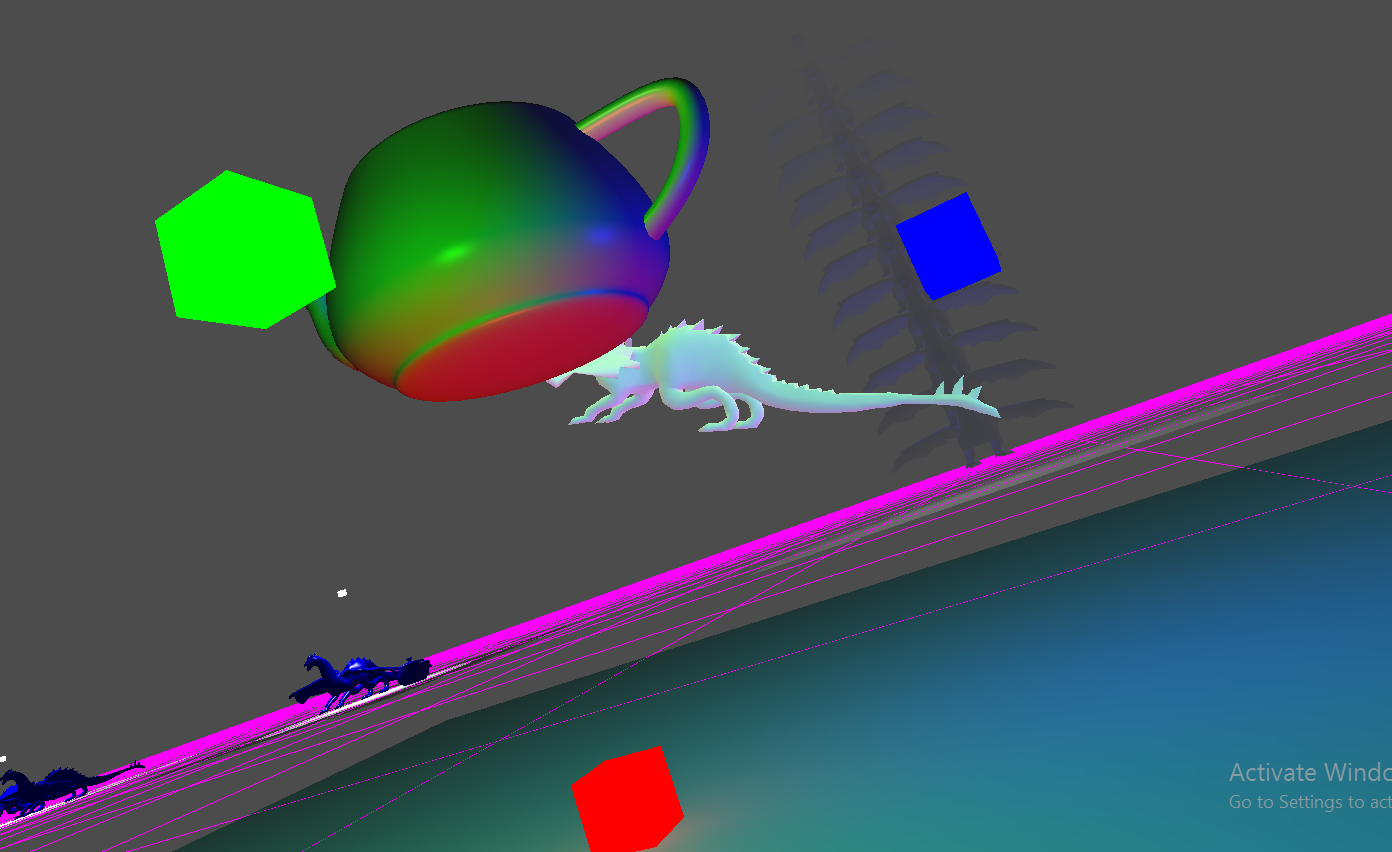
* **MULTIPLE LIGHTS SHADER SCREENSHOTS**
  + The first screenshot is a BIT of code (☺) showing that the multiple lights are hooked up using bits as stated in the lab doc. This code very simple starts or stops the correct lights.



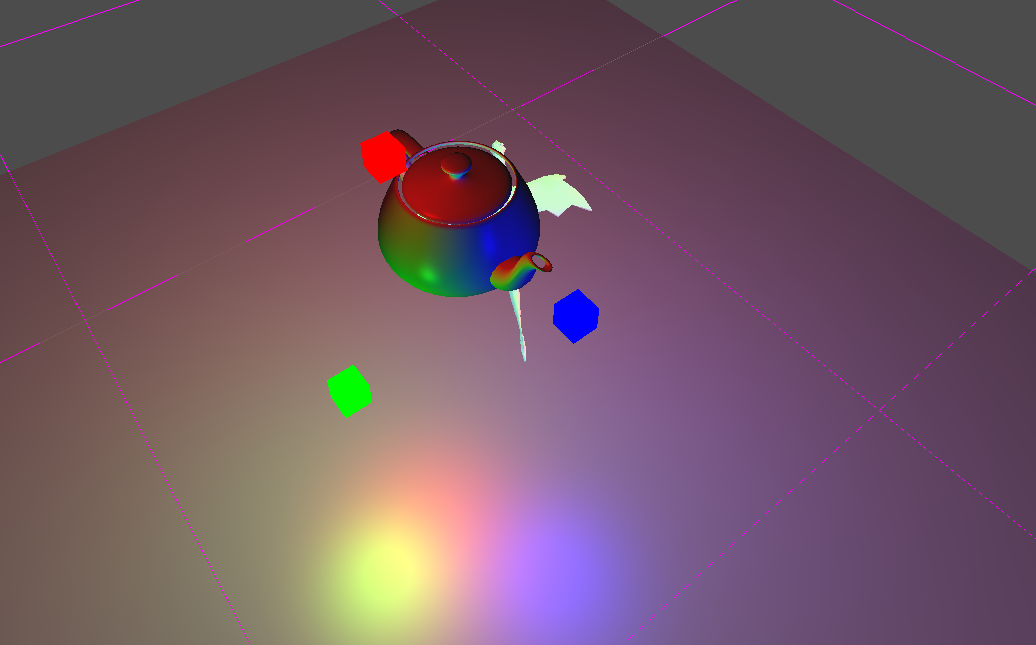
* + The next screenshot shows the Shader Parser output for the multiple light shader, which was used to get the name to use to query for the uniform location. It shows that the lights are setup as requested (struct + array).



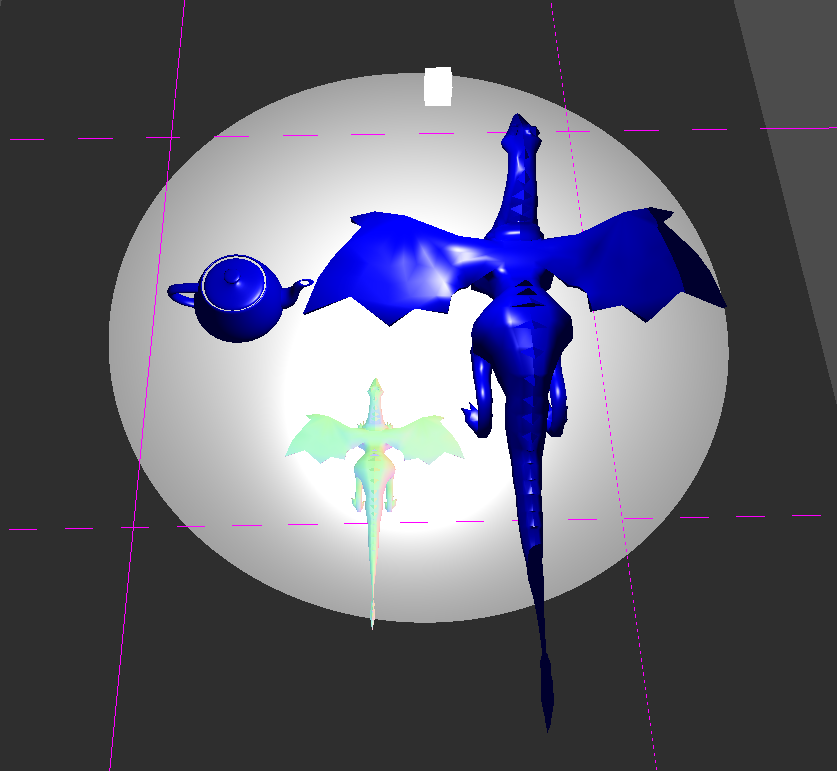
* + The next screenshot shows the teapot lit using multiple lights. One red light, one green light, and one blue light are used. It is easy to see all three of the colors reflected on the object, as it is affected by all three lights.



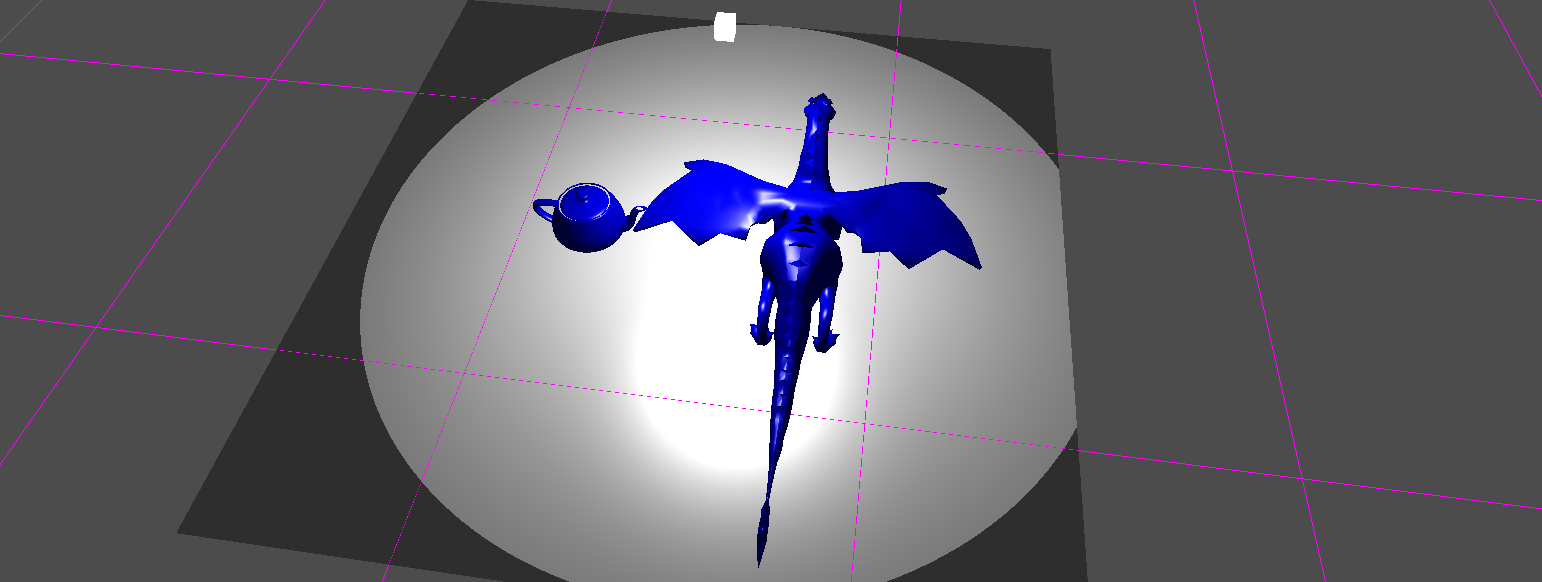
* + The next screenshot shows the object from a different perspective at a different point in time. This screenshot serves multiple purposes, the differing perspective allows the lights effects on the plane underneath to be seen, as well as shows that the lights all move independently and shows that the lights update on the teapot as they move.



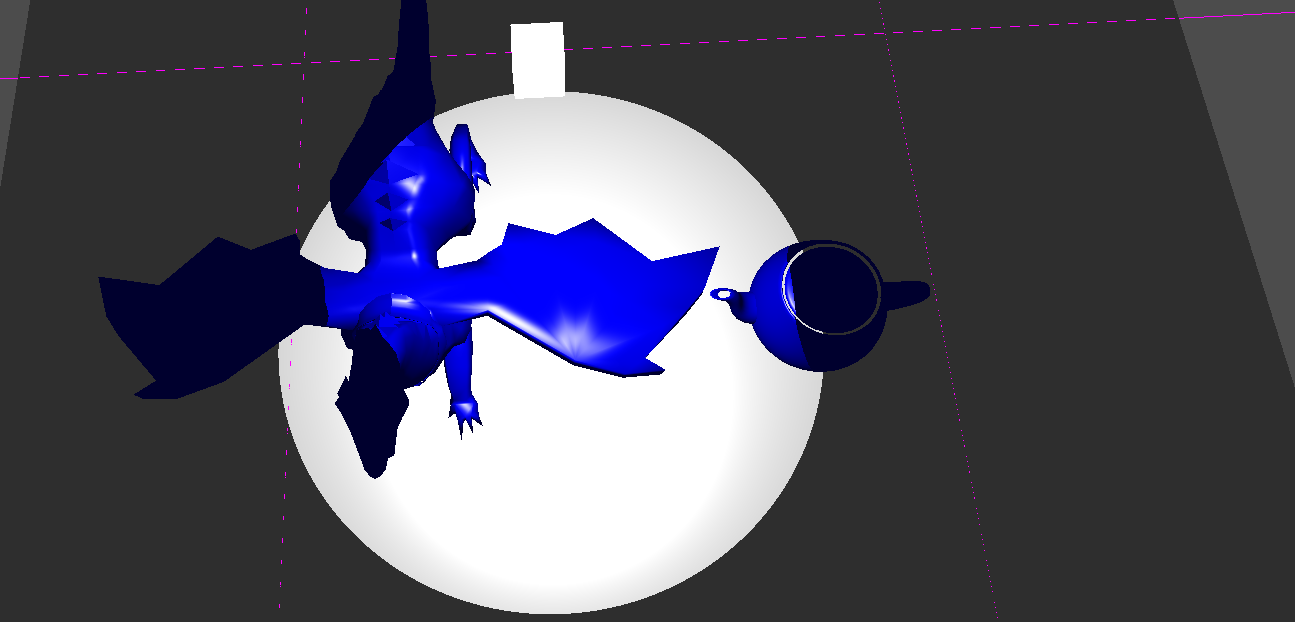
* **SPOTLIGHT SHADER DEMO SCREENSHOTS**
  + The first screenshot shows the spotlight directly about the teapot and blue BetterDargon facing straight down. Notice how the objects are lit with ambient, diffuse and specular within the cutoff angle, but only lit with ambient outside the cutoff angle.



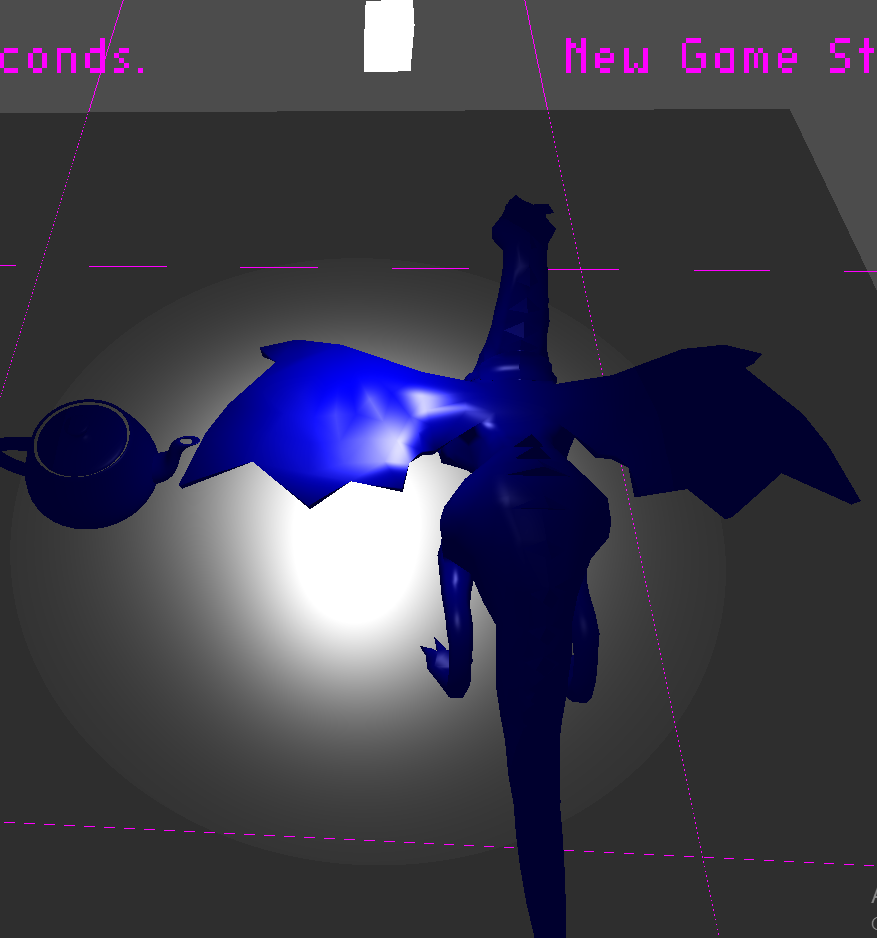
* + The second screenshot shows the same spotlight in the same position facing the same direction, lighting the same objects, with a larger cutoff angle. When comparing the two images, it is clear that the cutoff angle is working



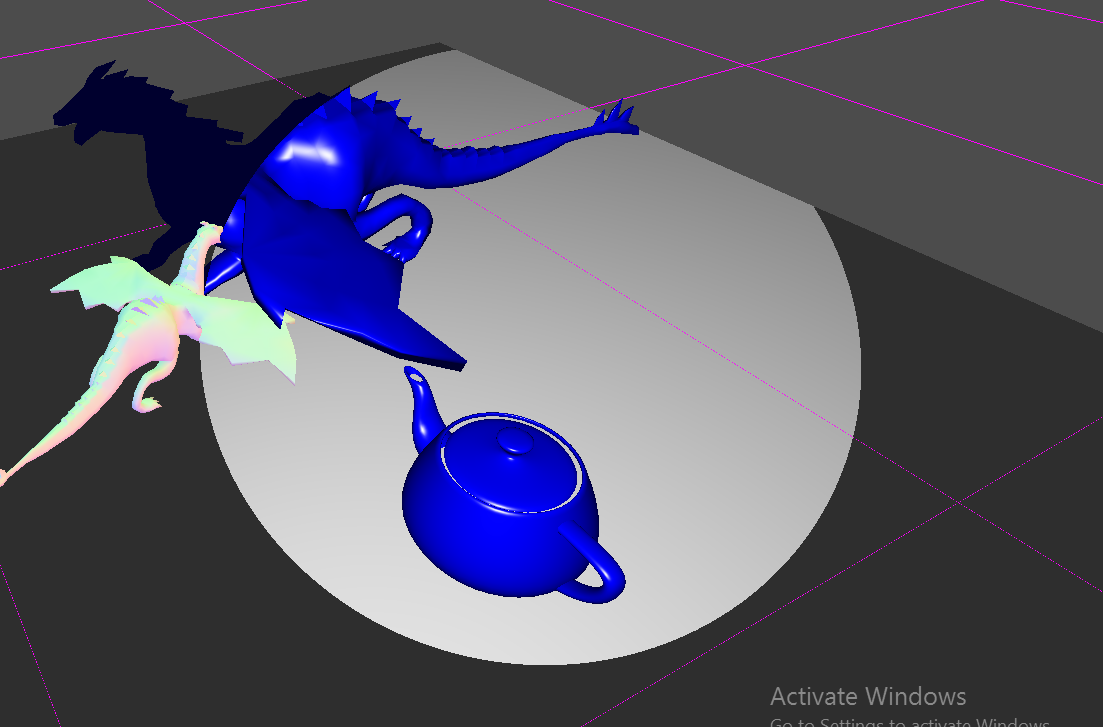
* + The third screenshot shows the same light, in the same position, facing the same direction, lighting the same objects, but with a smaller than default cutoff angle. Again, this demonstrates that the cutoff angle is working.



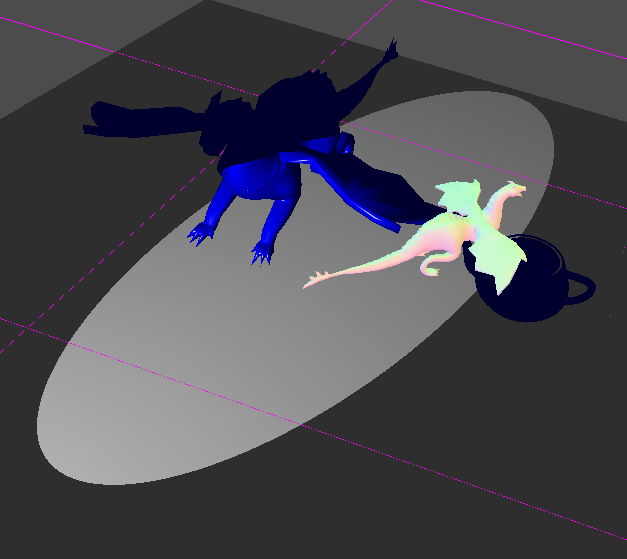
* + The fourth screenshot shows the spotlight with attenuation increased, revealing that the light does indeed get dimmer as it gets further from the center.



* + The fifth screenshot shows that the spotlight works at other positions and angles, as it has been moved with awesome mouse picking! This screenshot shows what it looks like while moving the light. In my opinion, this looks awesome, especially in real-time – it’s like you’re holding a flashlight or something. I could totally see this being part of a horror game or something (though perhaps the teapot would be scarier if it was blood-red)



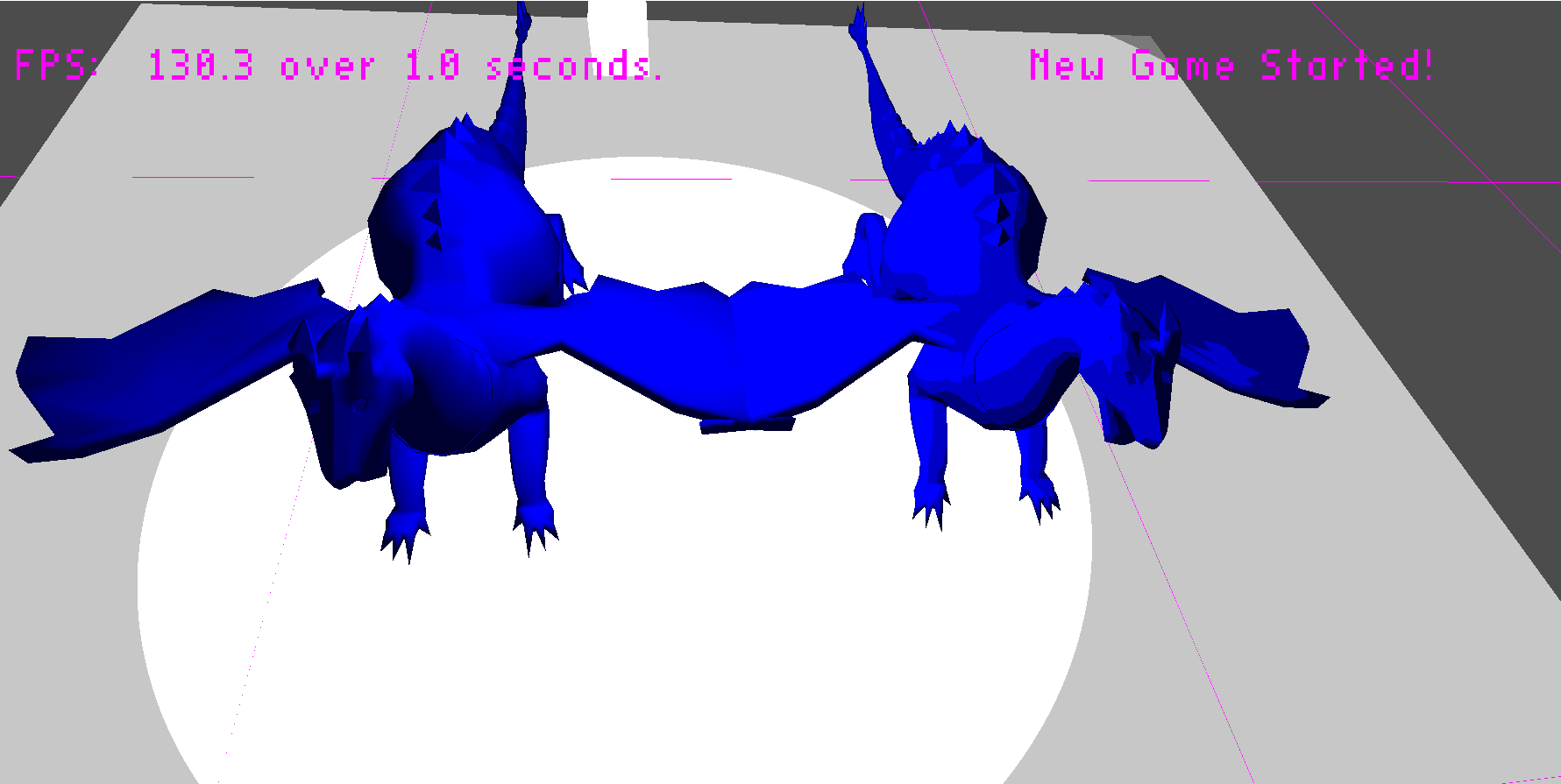
* + The sixth screenshot shows the light angle changed deliberately through use of my awesome mouse picking, the light set down, and the camera moved to show the ellipse effect created by the angled spotlight. Isn’t that cool too?



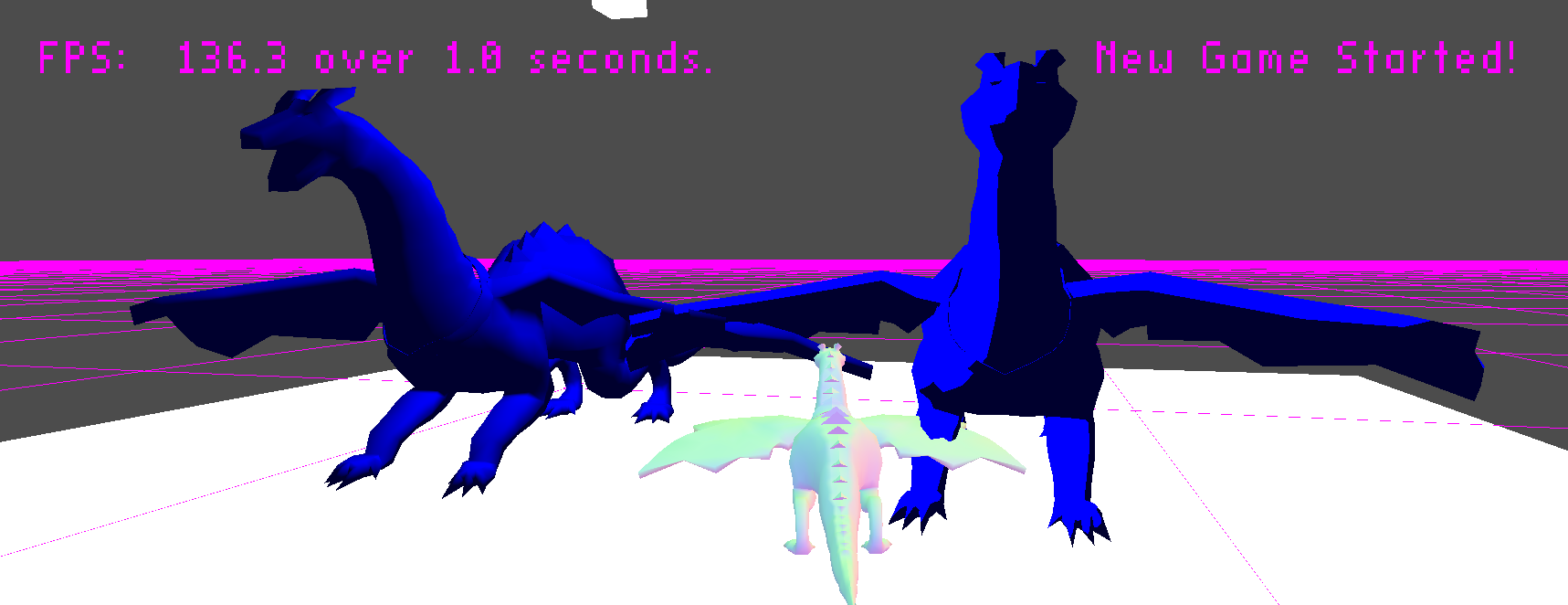
* **CEL SHADER SCREENSHOTS**
  + The first screenshot for the cel shader is yet another code screenshot, showing that the correct value for diffuse is quantized. Note that quantize is a function that exists elsewhere in my shader that I wrote myself. It uses math (or maybe magic? I forget…) to quantize a float and returns that float.



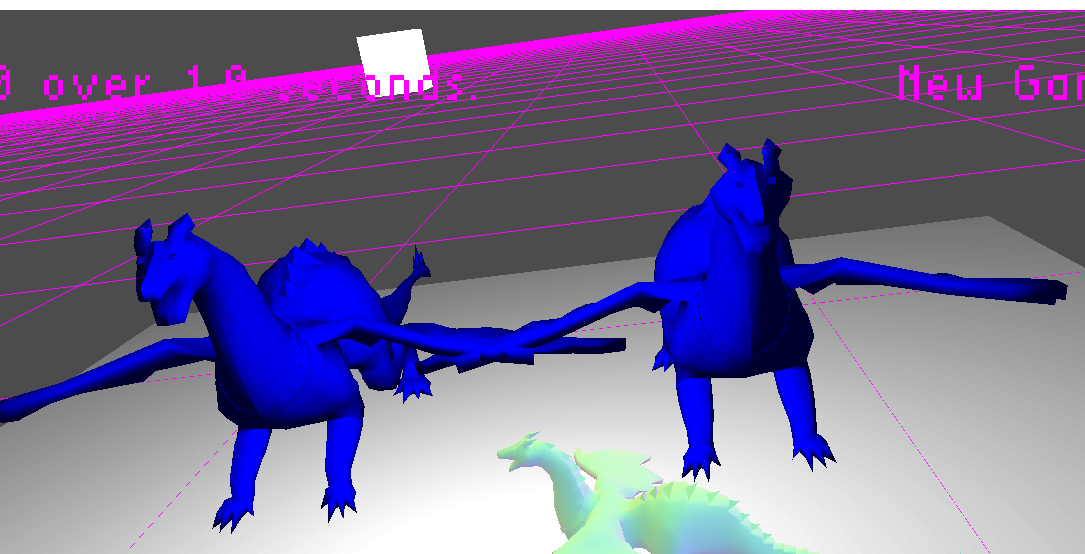
* + The second screenshot for the cel shader shows a dargon lit without its values quantized (to the left) and a dargon lit with its values quantized (right). Note that for this screenshot, numLevels = 4, and, there are four different light levels visible on the right dargon.



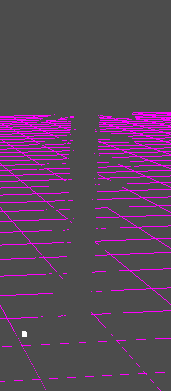
* + The third screenshot for the cel shader shows the same dargons, however, this time numLevels = 2, so that it is very clearly visible that the values are quantized properly. Notice there are only two different light strips on the right dargon.



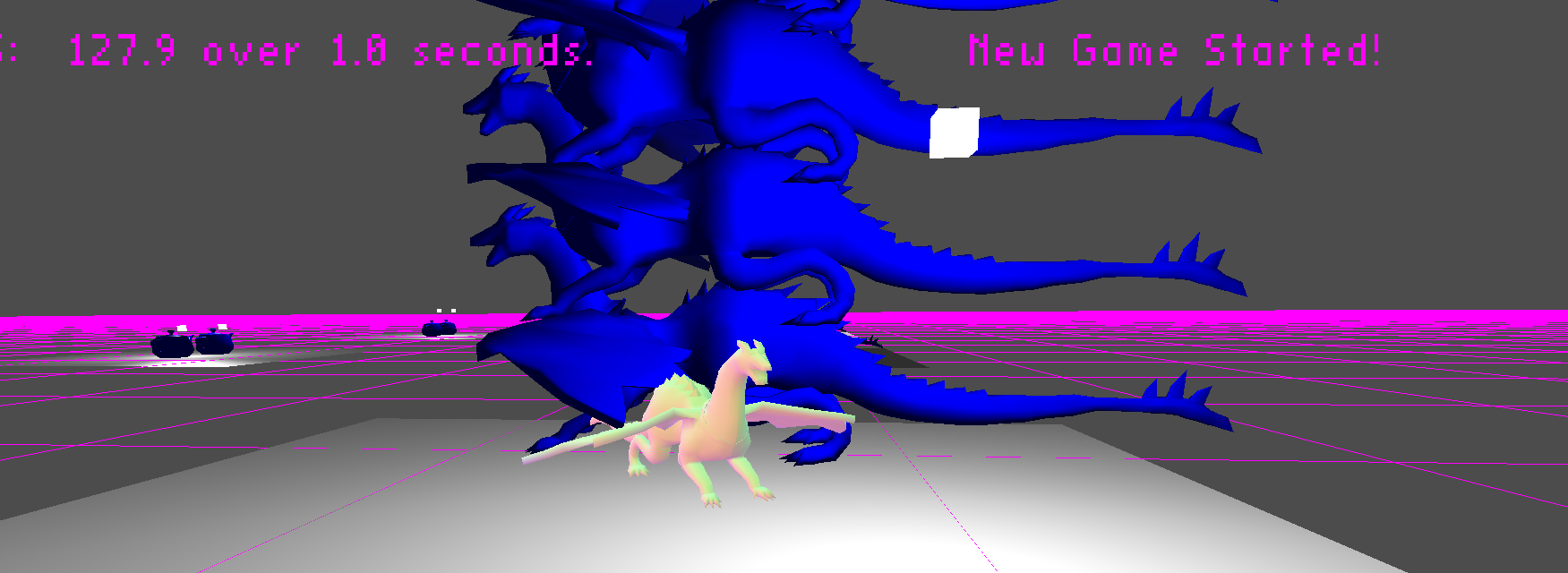
* + The fourth screenshot for the cel shader shows the same two dargons yet again, however, this time numLevels = 100. Note how the values are still quantized, but also note that the right dargon begins to look more like the left dargon because the light more closely mimics it



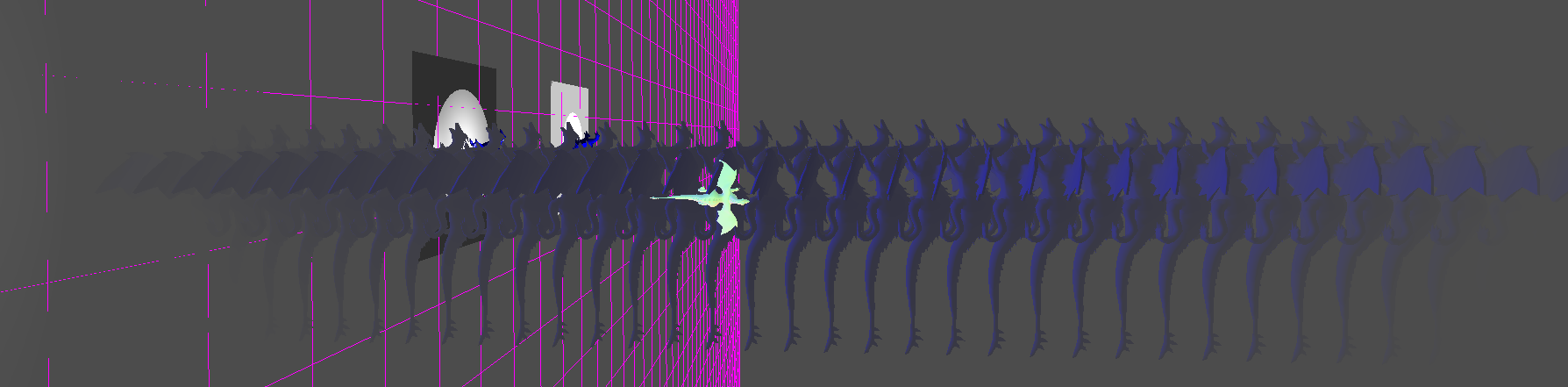
* **FOG SHADER SCREENSHOTS**
  + The first screenshot demonstrates the tower of better dargons at full fogginess. Note that they are above the maximum fog distance for this demo, and that they are only distinguishable because they cover up the grid (which is unaffected by fog). The fog color is the same as the background color for aesthetic purposes.



* + The second screenshot shows the bottom of BetterDargonTower in its maximum un-fogginess because all of the dargons and the plane are within the minimum fog distance.



* + The third screenshot shows the camera deliberately positioned to show off radial fog. Note how closer to the camera (near the middle) the dargons are more clearly visible than the dargons to either side (left or right, they are still further from the camera)



* + The fourth screenshot shows the simple linear fog calculation used in the shader



**Post-Mortem**

* I ran into a few errors, most of them were caused by my lack of following all of the rules of coding I should have been following while doing the lab. Most common of all was forgetting to update magic numbers, but some of my defensive logging definitely helped me catch those fast.
* I know I wrote bad code for this lab, because my game.cpp file gained like 500 lines, however, it’s not like I didn’t try to write good code. The very first thing I did was try to make a utility class that would allow me to treat the demos as groups – something that could let me specify the positions of the objects relative to the location of the scene they were a part of and what not. I actually spent a couple hours trying it, but when I got to a certain point I gave up on it because I couldn’t conceptualize how it would handle something – I’ve already forgotten the specifics, but I ended up deciding it would probably be easier to just have a method to set up each scene – which is what I did even though it resulted in tons of duplicate code. I know my code is ugly, and I would make it better if I had more time to dedicate to it, but I’ve already put over ten hours into this lab and I’m exhausted/tired of it. Also its due tonight and functioning, so why break stuff?
* Of all the shaders, I really enjoyed the multiple lights shader, the cel shader, and the spotlight shader. The spotlight, of course, being my favorite of the three. The rest of the shaders seemed quite redundant – we’d discussed phong and gouraud more than once before, directional lights vs. positional lights felt extremely trivial, and fog was just a copy-and-paste-and-add-a-few-lines type of shader, but the multiple lights shader was really cool – I enjoyed adding the bit testing code, making something look cool with multiple lights, etc. I also liked how for the spotlight and cel shader demos we handled the use of multiple keys at once (shift for 10x rate). I really liked using my mouse picking to better demonstrate the shaders in real time, I love the aesthetic of the cel shader and the spotlight shader – and I feel they were the most rewarding.
* I spent the most time setting up the scenes, actually. The shaders definitely ate up less of my time than placing teapots/dargons, running and testing, making sure my uniforms were set up properly, moving things, hooking up mouse input, etc. I’d guestimate the ratio of setup time to shader time as at least 3:1, which feels kind of wrong for a class that’s supposed to be more about shaders. Maybe it’s how I approached the setup, or maybe it’s my engine, but I’m positive that I spent way more time putting things places than writing shader code.
* Scope-wise, this lab was absolutely, without-a-doubt, bigger and harder than the previous lab. I’m not sure if I’d say this one was too hard, or the last one was too easy, but the difference is definitely significant and not negligible. I spent over ten hours on this lab, and not nearly that much on the previous lab, and I was surprised to hear others spent even more time than I did on this lab.