Lab 04 Textures

**Fufillment**

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

**Execution**

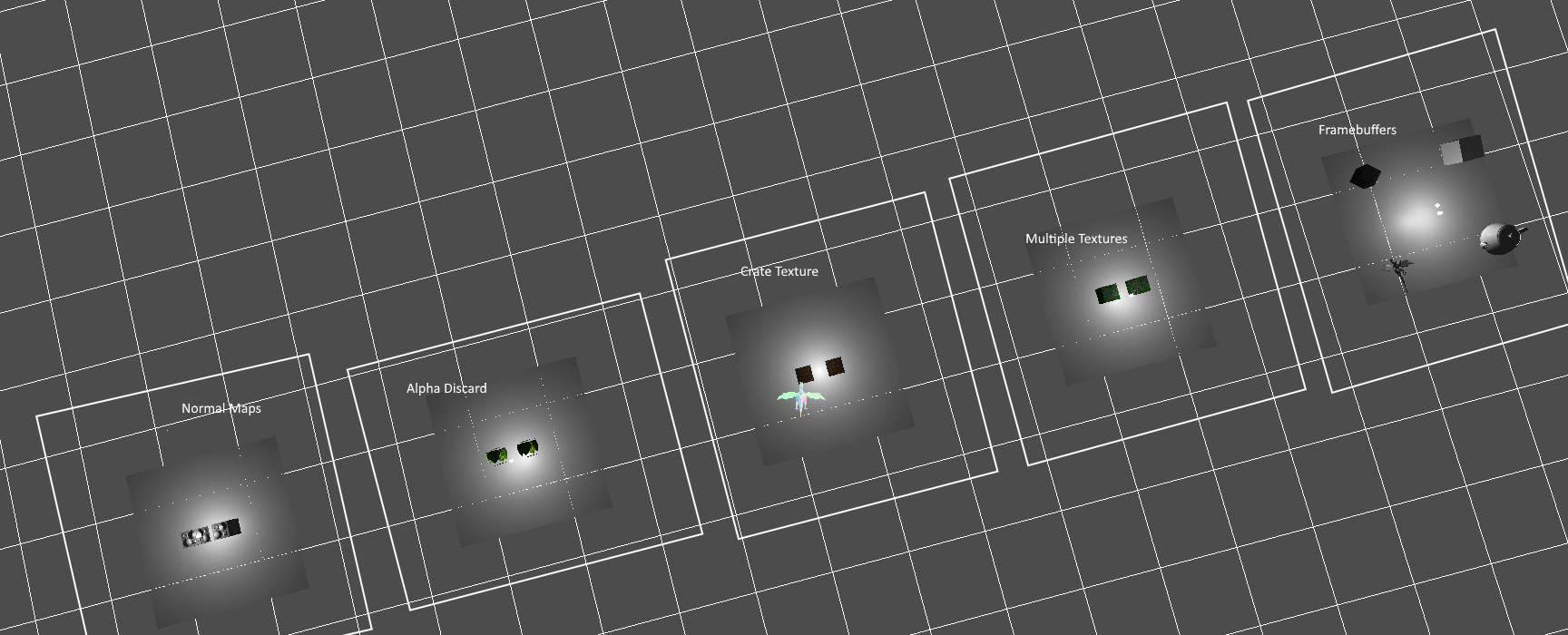
* I do not believe there should be any unexpected requirements for running

**Controls**

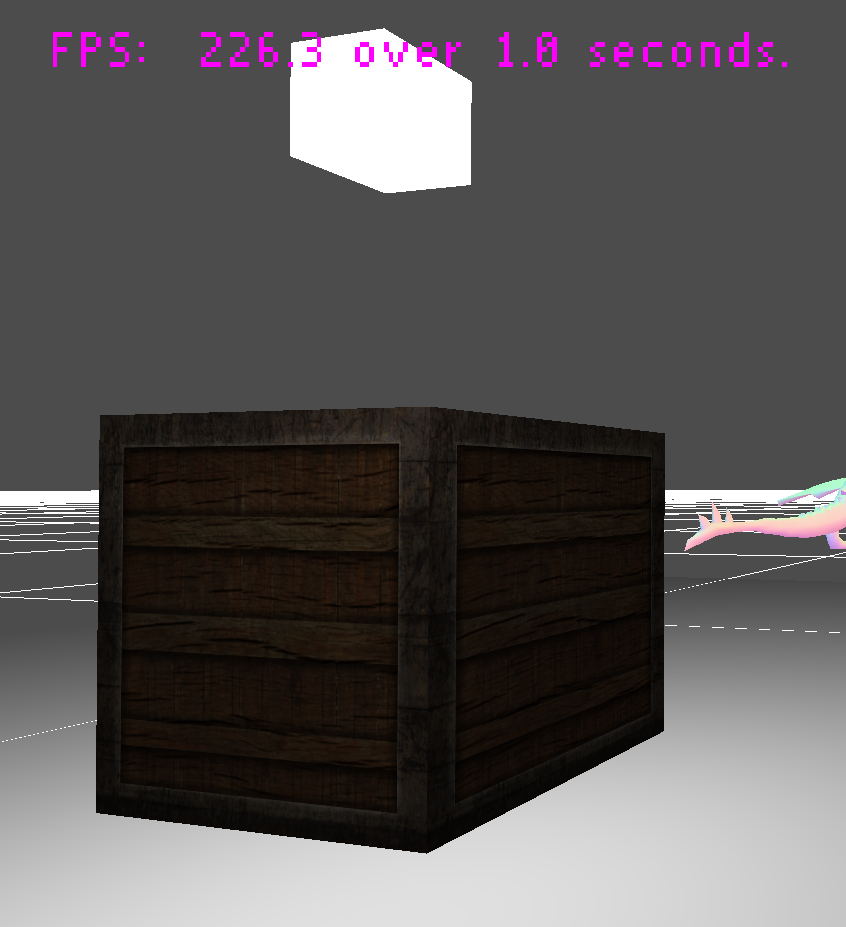
* Pressing the X key will close the application
* Pressing the P key will pause the application, pressing it again will un-pause the application
* Pressing numpad 0 will re-read the config file
* Pressing M, L, T or C will dump engine info to the console, this is pretty much exclusively used for debugging
* 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
* Lights can be moved via mouse input, clicking to select them and clicking again to de-select them, they will always be a certain height above the platform which is moused-over.

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

* **MISCELLANEOUS SCREENSHOTS**
  + Here we have a general overview shot displaying the layout for all of the shader demonstration areas for this lab. The annotations have been manually added using paint.net. It’s a good thing dargon can fly, otherwise it would have been hard to get into the air to take this screenshot!

****

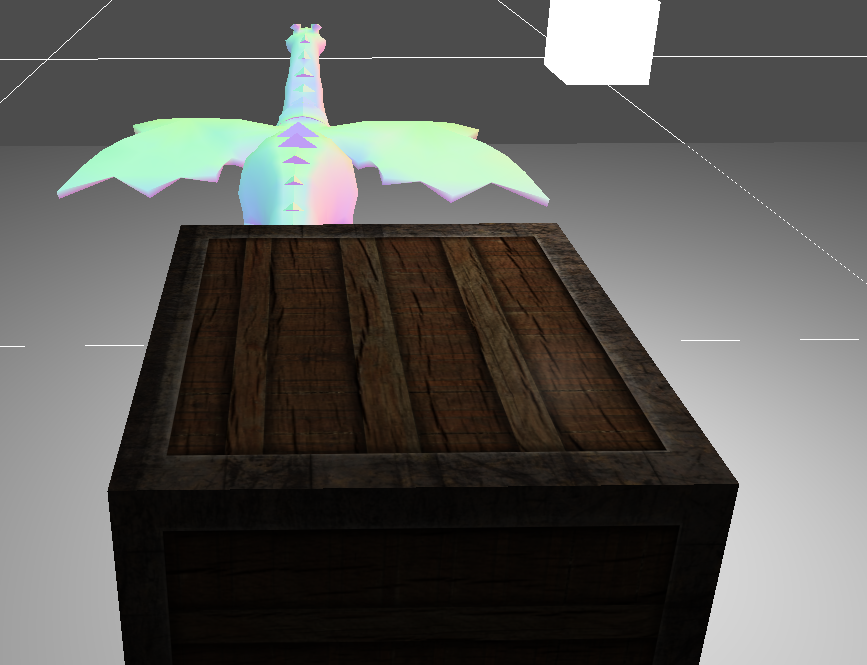
* **SINGLE TEXTURE SCREENSHOTS**
  + Shows a simple cube with position texture and normal data using a shader that applies lighting and the texture. The light and the crate can both be seen in this image. Though the angle may not show it, the light is indeed in front of the crate, between the crate and the camera, thus lighting the side of the crate that can be seen.



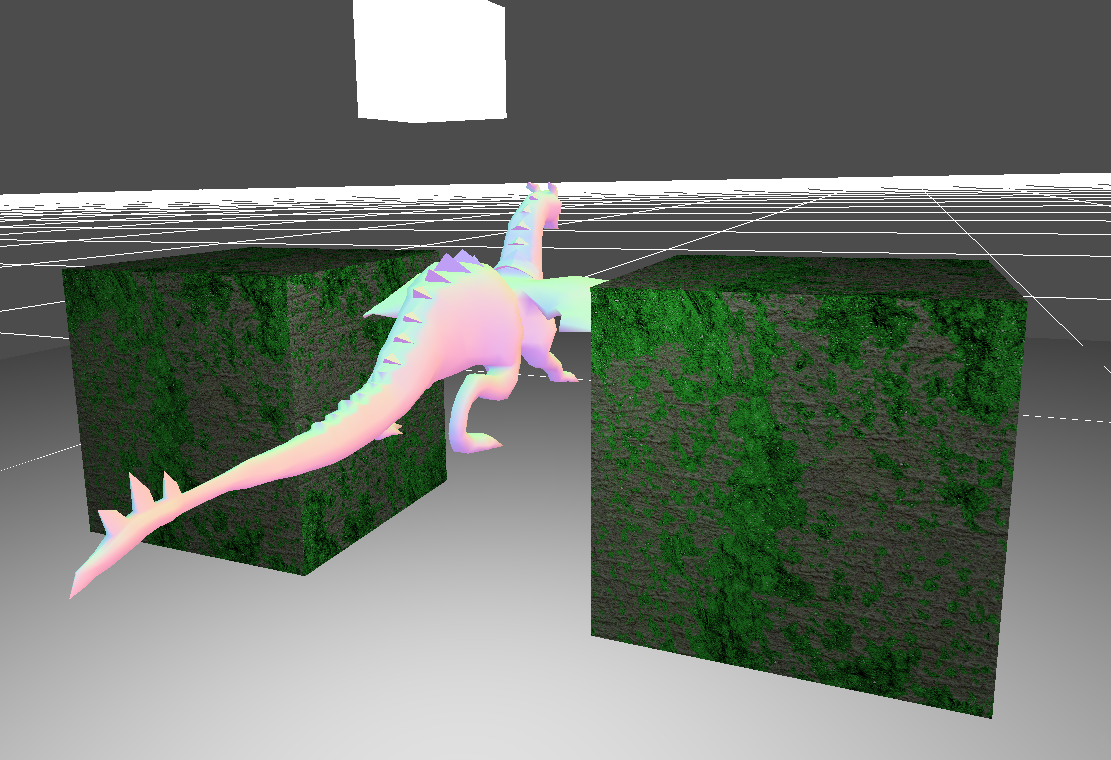
* + The following screenshot shows that the texture color is multiplied by the ambient and diffuse light, with specular light added separately. This is the same math, (though written a little differently) when compared to the light equation we were told to use for this shader.



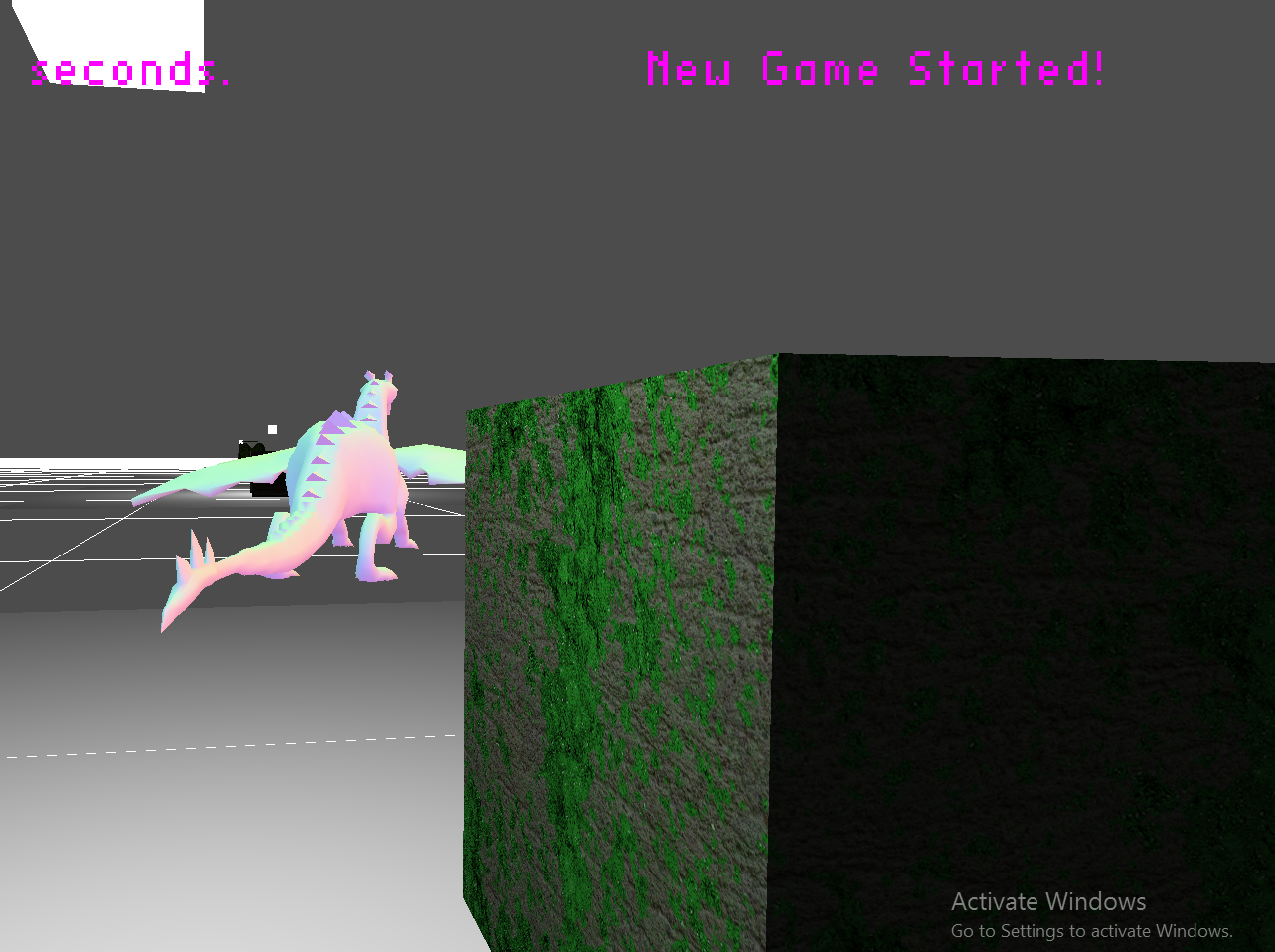
* + This screenshot shows that there is indeed specular light affecting the crate using this shader, and it also shows a darkened side of the cube, on which the ambient light can be seen more effectively. Finally, it shows a different perspective of the cube, indicating that it is indeed textured on all sides.



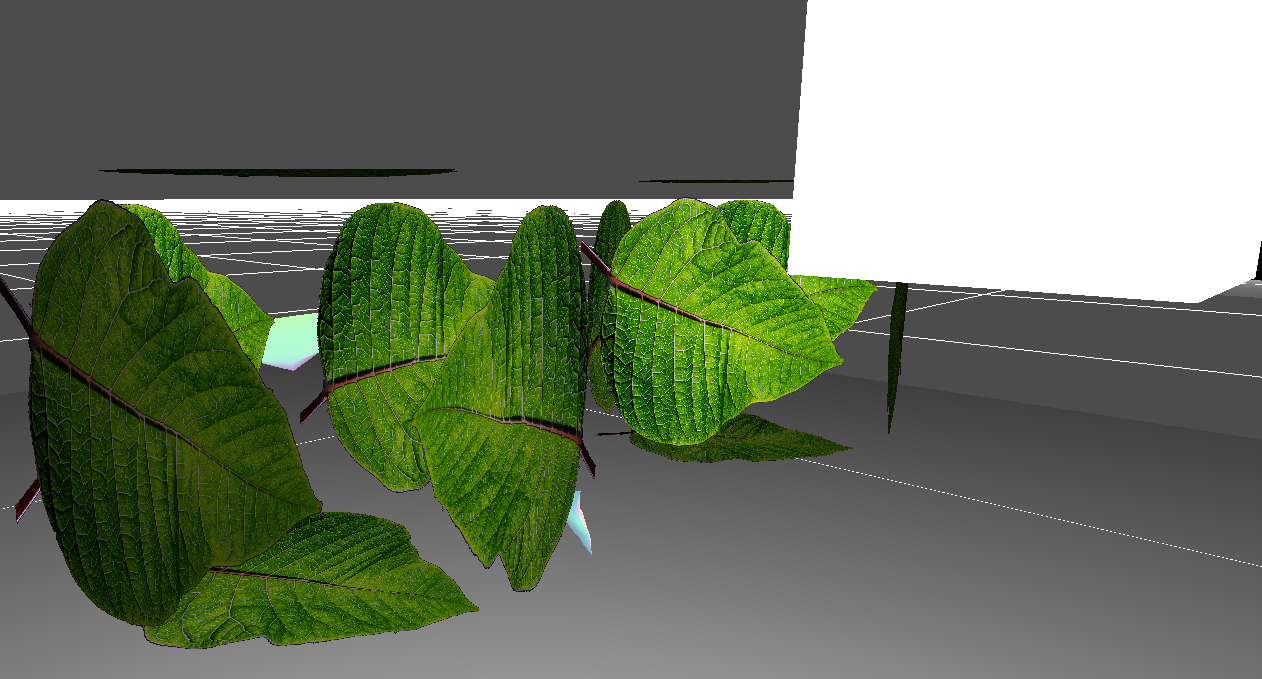
* **MULTIPLE TEXTURES SCREENSHOTS**
  + This screenshot shows multiple textures implemented effectively on two cubes. They are using the provided cement texture and have a moss texture (with transparency) overlaid on top of that texture, giving the final appearance of mossy cement.



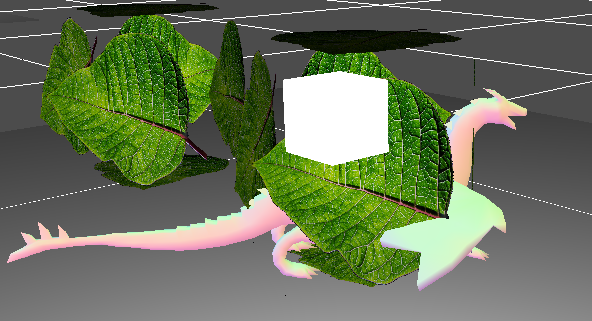
* + This screenshot shows a different perspective of one of the cubes using multiple textures. In this perspective, the effects of the light on the cube can be more easily seen.



* **ALPHA DISCARD SCREENSHOTS**
  + The alpha discard shader is used on the objects in the following screenshot. The leaves are applied to a cube with culling disabled, so that both sides of the leaves can be seen. Note how you can see dargon between the leaves, as it shows that the pixels are discarded. Also note how the leaves are lit.



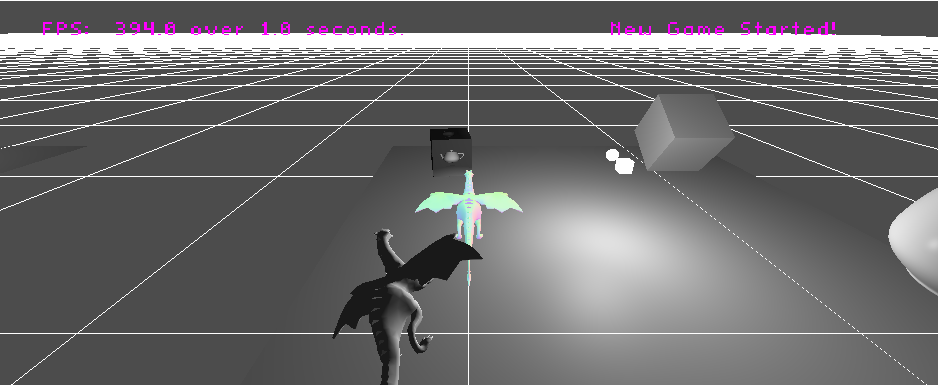
* + This screenshot really tries hard to show off the discarding aspect of the shader by placing dargon inside of the cube. He can be seen everywhere outside the leaves, indicating that pixels are indeed discarded based on the mask texture used.



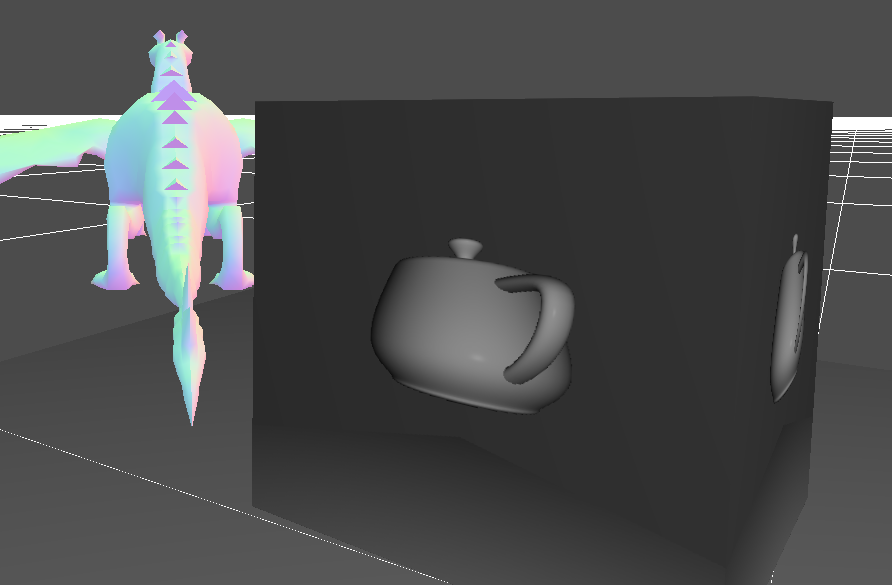
* + This screenshot gives shows the leave image used as well as the mask associated with it for context.



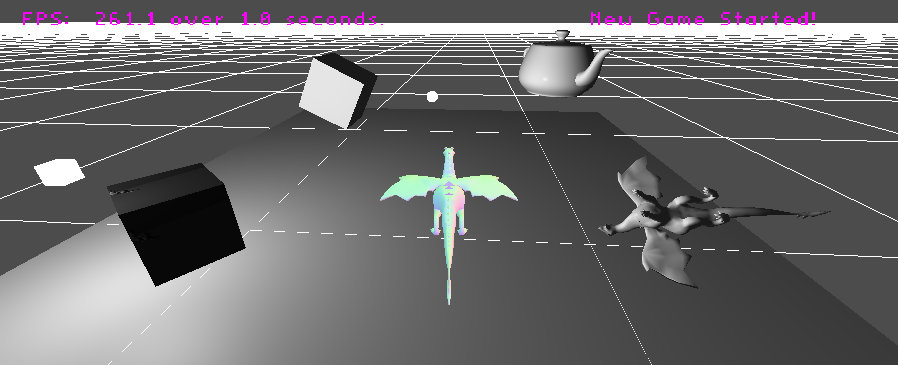
* **FRAMEBUFFER SCREENSHOTS**
  + This screenshot shows the application on startup, at which time the framebuffer cube is, as requested, visible onscreen.



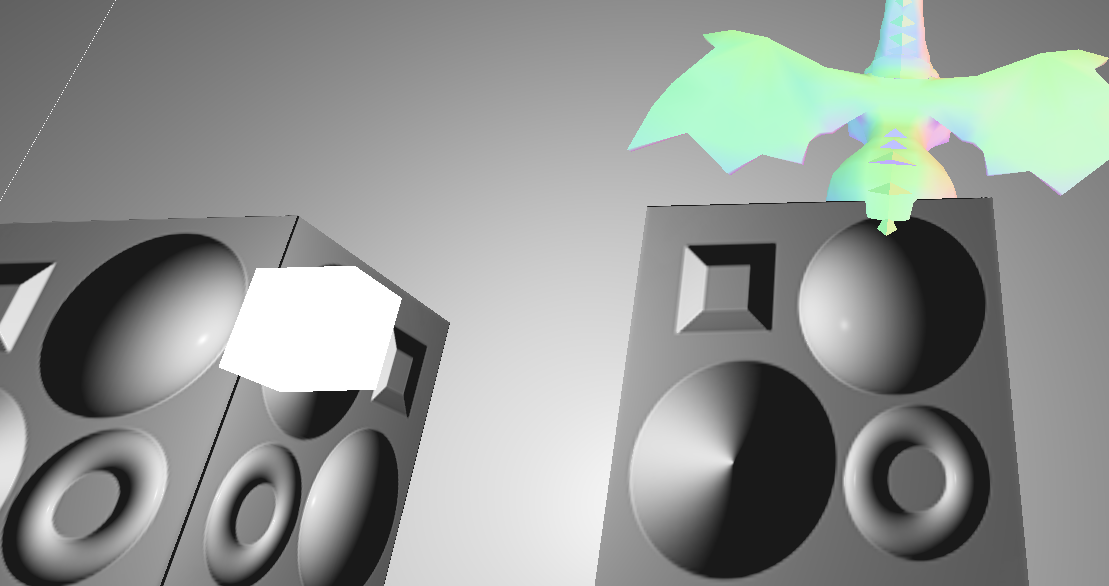
* + This screenshot shows a close-up of the framebuffer cube while the camera is viewing the rotating teapot. Notice that the teapot is lit properly. Small specular highlights can be seen, as well as the diffuse tapering off near the left edge.



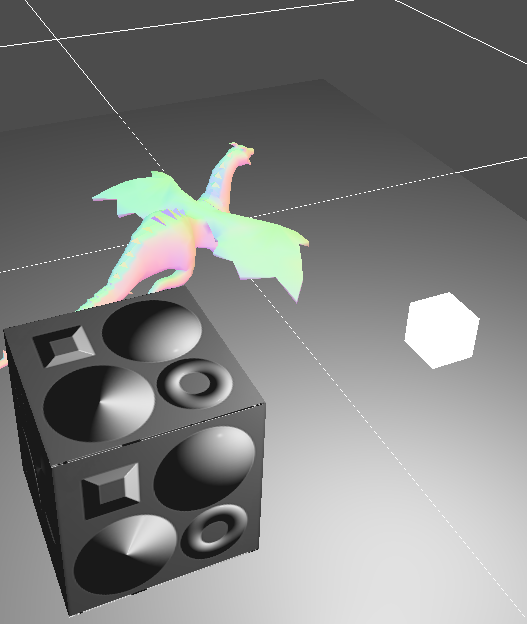
* + This screenshot shows the scene that the camera for framebuffers is viewing. The camera rotates slowly so that at different times the cube, teapot, or dargon can be seen spinning. The camera is the white orb located in the center of the platform.



* **SOMETHING COOL (NORMAL MAPS) SCREENSHOTS**
  + The first screenshot shows two cubes using the normal map shader I wrote as something cool. Notice how there are ambient, diffuse and specular components, as well as how the lighting lights the normals in an appropriate way. The cube surface is indeed flat, but the normals and the way they are affected by light gives it a much more 3D appearance.



* + In this screenshot the light is deliberately moved to a different location in order to show that the normal mapping does indeed work. The object correctly responds to the new position of the light.



* + Shows the normal map used, for context.



**Post-Mortem**

* I ran into a few errors, this time not caused by bad coding practices. Two in particular really stand out:
  + The first error that got me was that upon trying to read in a leaf texture unmodified, my game engine crashed. This really surprised me, firstly because I was highly confident in the capabilities of my bitmap loader, and secondly because there was no good way to debug it whatsoever! The exception was a read access violation, but it occurred outside of my own code. When I went to the call stack, not a single method I had written was there, so I couldn’t see what was throwing the exception! I was dumbfounded, so I popped a breakpoint in initialize, and stepped through my engine and game one line at a time all the way until I found the line that caused the crash, which was very annoying and time consuming. Upon finding the line, I googled it with the word crash, and found a few websites that were not helpful whatsoever. However, upon going to the fifth or sixth site I visited, I found a section on “commonly made OpenGL errors” or something similar. In that section, near the beginning of the textures subsection, I found something that claimed that textures in OpenGL must have a width divisible by four because of the way OpenGL handles them internally, unless you put another method call in there to explicitly tell OpenGL that your texture’s width is not divisible by four. I was suspicious at first, but without touching a single line of code, I re-sized the texture, ran my engine, and it worked perfectly, so apparently the website was correct. This is a really unexpected and hidden thing to keep an eye out for in the future!
  + The second error I made was forgetting to clear the depth buffer bit after binding my framebuffer. This one also stood out for being exceptionally hard to debug because all I was getting was a black cube. There were no exceptions, no errors logged (and I log often…) and no OpenGL errors found! I stared at the code you had given us for quite a while, trying to see if I had forgotten anything, but eventually I deemed that I had not and started looking at other things, like perhaps I had the camera pointing at nothing (which was not the case). After hours of looking, I asked someone who had done framebuffers before, and after they asked a couple of other questions, they asked me to check this one, which was sure enough the case. Man, I really despise those issues that give no useful debug information.
* When it comes to which of the sections of this assignment was my favorite, it would definitely have to be the framebuffers. The reason for this is two-fold: first, I did not have time to over-achieve with my “something cool” shader, which would normally be my favorite for the simple reason of it being my own, second, I can see tons of potential for framebuffers. I can imagine the uses of being able to show a different part of the game world in another location (security camera, reflection, etc.) and I am really excited to know that my engine is now more powerful because of this reasonably-sized addition.
* For this lab, more of my time went into debugging those two issues I ran into than actually setting up the scenes or writing the shaders because I had the general structure already completed from the previous assignment. It was as easy as re-name, re-organize, re-place shader, which made this lab much more fun to do because I felt like I was making progress more of the time as opposed to doing mindless setup. Had I not ran into those two aforementioned issues, this lab would have actually been really enjoyable for me. The ratio of time required to cool things would have been more than acceptable.
* Scope-wise, this lab was pretty appropriately sized. It was smaller than the last lab (which consumed my entire weekend) while still being large enough for me to feel like I learned things about textures, framebuffers, engine structure and shader writing. I liked this lab, and I only wish that I had more time to work on the “something cool” shader –because I totally would have dumped hours into making some sort of game out of it if it weren’t for other classes and miscellaneous responsibilities slowly eating away all of my days.