CSCI 441 - Lab 05
Friday, September 30, 2022
LAB IS DUE BY **FRIDAY OCTOBER 07 2022 11:59 PM**!!

Today, we'll be enhancing our flight simulator visually by adding lighting around our scene.

Please note:
- We will not be using the specular or ambient components in this lab. We will only be using the diffuse component.
- We will be implementing only a directional light.
- We will be implementing Gourad Shading.

For future assignments/projects, you will need to implement all three lighting components (diffuse, specular, ambient) as well as all three light types (directional, point, spot) with attenuation as appropriate using Phong Shading.

Please answer the questions as you go inside your README.txt file.

Step 0 – New Library Files

You'll notice an include folder with this lab. Copy the files into your `Z:/CSCI441/include/CSCI441` folder as this gives us some additional helper classes and updated functionality.

Also, in CLion be sure to update the working directory to be the parent (Run > Edit Configurations > Working Directory = ..).

Step 1 – Light Up the Buildings!

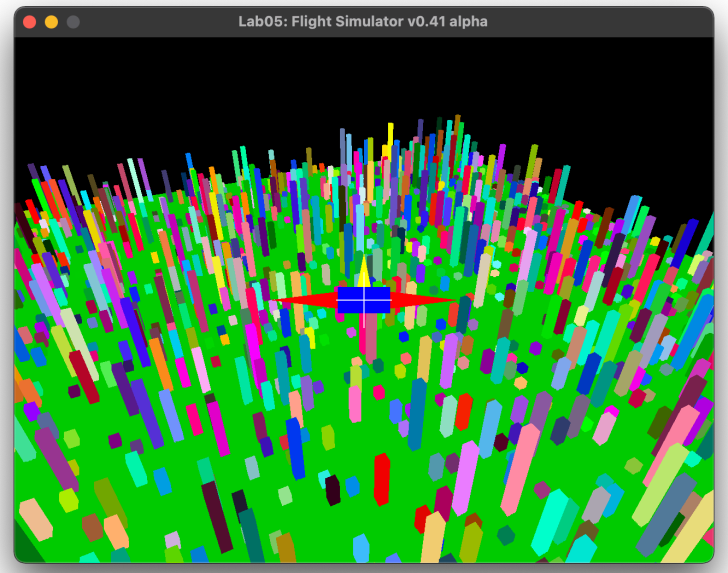
Start off by running the existing code. There's a lot of starter code already in place for you:

- A working free cam
- A working Shader Program (take note of the new `CSCI441::ShaderProgram` class to wrap all of our shader registration steps)
- A working VAO for the ground
- A hierarchically structured Plane object

We'll be modifying several pieces of the above to implement our lighting.

Once running, you can press space to fly through the world and use WASD or the mouse to turn. Press shift to fly in reverse.

How's everything look? Pretty bright?



Currently, we are setting the color of every fragment to be the same and we are seeing these flat shaded objects that have no depth or distinction to them.

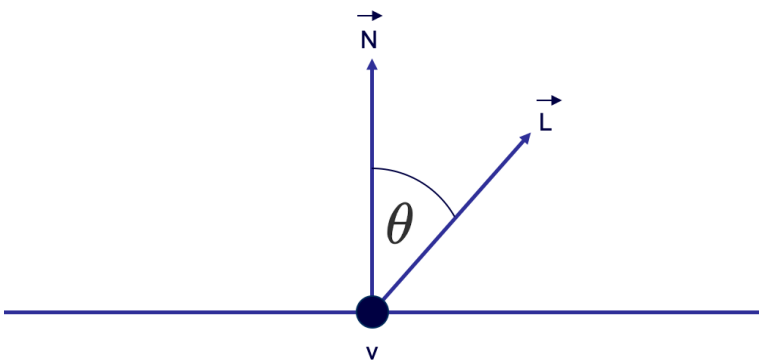
We are going to implement a diffuse directional light to illuminate everything a bit more realistically. This will be done in two parts (1) the actual lighting calculations in the shader on the GPU (2) sending all the specific parameter values to the GPU from the CPU.

Part I – Update the Shader

Everything we're about to do will occur in the vertex shader, in `shaders/lab05.v.glsl`.

If we look at what is currently happening, we are
(1) Outputting the vertex position in clip space
(2) Outputting our color varying to be the material color
It's this second step we'll need to modify the color that we are assigning to our varying.

As a reminder, the diffuse component is only dependent upon the vertex normal and the direction to the light.

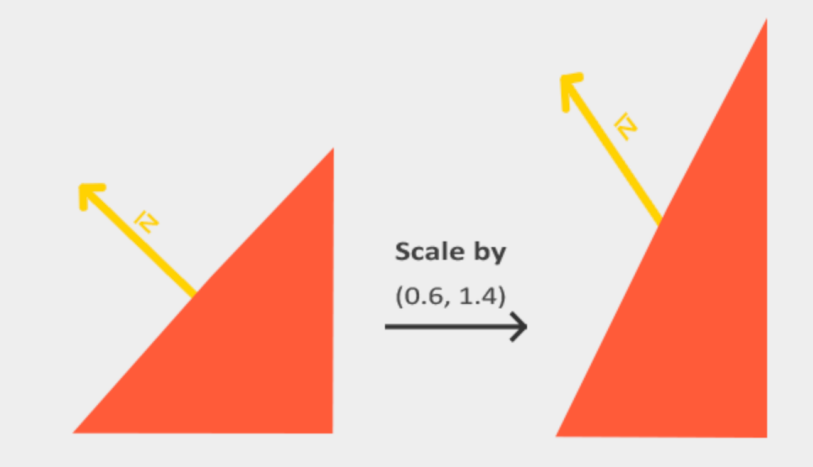


Let's go through the process and calculations in order. First, we will need to know the direction the light is coming from and the color of our light. Add these as uniforms at `TODO #A`. Both will be represented as a `vec3`. As far as the shader is concerned, we don't care what these specific values are (we'll specify them in Part II on the CPU).

We can now start the calculations for the diffuse property. The light direction that we specify with the uniform represents the direction that the light is traveling. However, for our calculations our convention has the light vector pointing at where the light is coming from. Therefore, we'll first need to reverse the direction of our light direction uniform and then normalize this value. All of our uniforms and attributes come in as constants, so we'll need to create a new variable to store the result of this calculation. Perform this step at `TODO #B`.

We now need to get our vertex normal. This will be an attribute that is specific to each vertex, so add another attribute at `TODO #C`.

Now, unfortunately, we can't just use the vertex normal as it exists directly. When we specify the vertex normals for each vertex, we are specifying them in object space. When we perform the lighting calculations, we need to ensure that all of our vectors exist in the same coordinate space. The light vector exists in world space, so we'll need to transform the vertex normal into world space as well. Well, unfortunately again, we can't just multiply the vertex normal by the model matrix. The reason we can't is due to non-uniform scaling. If we were scaling by a uniform amount in all directions equally, then only the vector's magnitude would change – not its direction. But when each direction is scaled a different amount, then the vector's direction is changed and our normal would no longer be normal. The following 2D diagram demonstrates this effect.



Luckily (phew) there is a solution to this problem and that is the **normal matrix**. The normal matrix uses some linear algebra magic to ensure after scaling that our normal remains normal. The result of all the magic is to set the normal matrix to be the upper 3x3 matrix of the transpose of the inverse of our model matrix. We only want the upper 3x3 so we remove the translation component and it makes the math happy to multiply a vec3 by a mat3. Don't worry about how to make the normal matrix for now, we'll handle it on the CPU side.

(If you really want to know how the matrix is composed, this article walks through the math – but uses the legacy GLSL code. The math is still correct. http://www.lighthouse3d.com/tutorials/glsl-tutorial/the-normal-matrix/)

What we do need to do in the shader is to create a uniform to receive this matrix. At `TODO #D`, add a `mat3` uniform for the normal matrix and transform the vertex normal at `TODO #E`.

Great! We now have our light vector and normal vector both in world space. We're ready to do the diffuse calculation. The diffuse component is calculated by the following equation

$$I_d = C_{LD} * C_{MD} * max(\vec{L_w} \cdot \vec{N_w}, 0)$$

We multiply the diffuse color of our light by the diffuse color of our material by the max of the dot product between our light and normal vectors and zero. The max zero prevents the backside of our triangles from being lit. Do this calculation at `TODO #F`.

Finally, update `TODO #G` to set the color varying to be the result of our diffuse calculation.

That's it. All of the lighting calculations are complete. However, if you were to run your program again you'll just see….black. All those additional uniforms and attributes the shader requires as input do not currently have any values. To the CPU!

Part II – Update the OpenGL Code

It's now time to set all of the inputs to the shader so our calculations will work. We'll need to do this in a few places within our `Lab05Engine` class. Let's start with the bookkeeping.

At `TODO #1` and `TODO #2`, add variables that will ultimately store the locations of the new uniforms and attributes we created in our shaders. And now at `TODO #3`, we'll assign these values. Follow the same style as what is already started. Perfect, we've queried the shader for the locations of these values – now we'll start sending the data over to the GPU.

Going in order of our program execution:

Part II.A – `Lab05Engine::_setupBuffers()`

We'll first hook up the vertex normal attribute with our CSCI441 object library (used for drawing cubes, cones, etc.) At `TODO #4` we have the entry point to our library and currently only providing the vertex position attribute location. We can pass a second argument corresponding to the vertex normal attribute location. (And we can actually pass a third argument that we will do next week).

Next we need to hook up our plane to the normal matrix uniform (we'll calculate it's value later, stay tuned). At `TODO #5` we are currently passing -1 as the normal matrix location. Provide as the second argument the normal matrix uniform location.

Part II.B – `Lab05Engine::_setupScene()`

We'll now set up our light information. The light never moves or changes so we only need to set this once. Find `TODO #6` in `_setupScene()`. Create two vec3 variables to store the light's direction and color. Set the direction to be (-1, -1, -1) and the color to be white (1, 1, 1). We're now ready to send over

the uniform information. Use the function `glProgramUniform3fv()` to send over a vector of three floats. The arguments to this function are:

1. The program handle the uniform corresponds to (accessible via `_lightingShaderProgram->getShaderProgramHandle()`)
2. The location of the uniform to send the data to
3. The number of three element vectors we are sending (we're sending one at a time)
4. A pointer to the start of the data in CPU memory. The way to access this using the glm library is as follows

```
glm::vec3 exampleVec;    // create a vec3
&exampleVec[0] // the address of the first element of the array
```

Do this for each of our two light uniforms.

Part II.C – `Lab05Engine::_computeAndSendMatrixUniforms()`

There's only one last shader input we haven't sent yet – the normal matrix. The dreaded linear algebra dark arts we need to calculate. Don't fear, glm has us covered. At `TODO #7` we are precomputing all of the matrices CPU side and then sending them to the GPU. While sounding complex – "the upper three-by-three of the transpose of the inverse of the model matrix" – it's simple to write in code. This equation

$$N_{3x3} = ((M_{4x4}^{-1})^T)_{3x3}$$

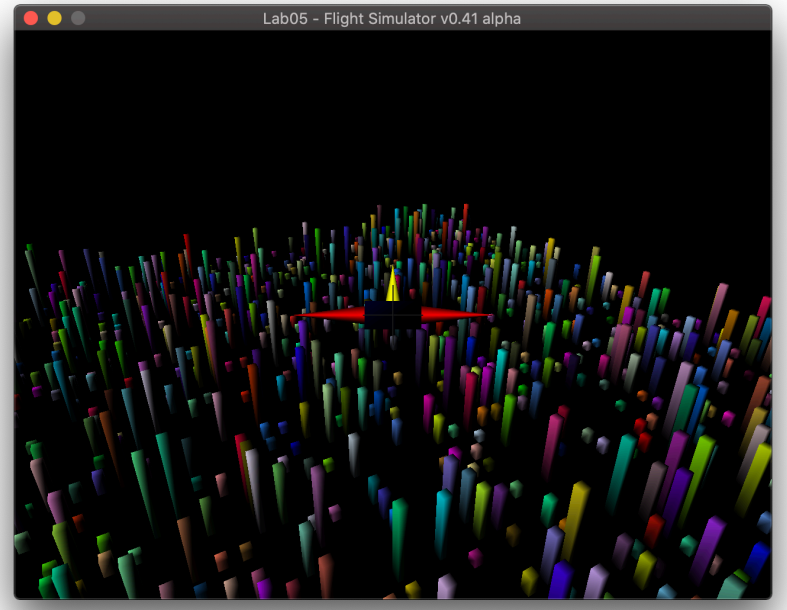
Translates to the following code leveraging glm

```
glm::mat3 normalMtx = glm::mat3(glm::transpose(glm::inverse(modelMtx)));
```

Now send the normal matrix to the GPU the same way the MVP Matrix is getting sent. Note how the library class abstracts and simplifies the process we were doing for the lights.

If you go and check out the `Plane::_computeAndSendMatrixUniforms()` method, you'll see the normal matrix calculation and uniform sending already in place for you.

We've satisfied all of our shader inputs. Hooray! Run the program and check out our nifty looking buildings and plane!



Wait a minute…

We lost our ground. Boo-urns.

Step 2 – Creating a Floor

All of our CSCI441 objects are specified with vertex positions and vertex normals. Using the class library hooked the data up to the shader properly. But our ground is being manually created and has no normal data specified. Let's add that information in so we once again have a floor.

The `_createGroundBuffers()` function is creating our ground VAO. It needs some more data to interact with the shader. Since the boilerplate is already in place, we only need to expand its definition.

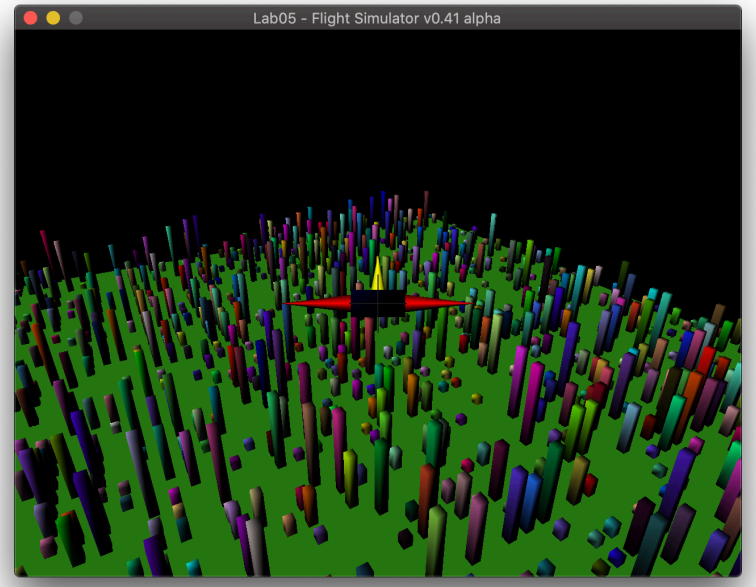
At `TODO #8`, add values to the struct to track the (x, y, z) values of our vertex normal.

Now at `TODO #9` when we specify the actual vertex data, set the vertex normal. Since our ground lies in the XZ-plane, all of the normal should be aligned with the positive Y-axis.

Lastly at `TODO #10`, repeat the steps that were performed to connect the vertex position.

1. Enable the attribute location
2. Set the pointer. The data is interleaved so the normal start after the size of three floats.

We can now rerun the program one final time and see our ground reappear!



Fly around the world and take note of how the directional light works and observe the local illumination. No building is casting a shadow on another or on the ground (local illumination). The sides of the building facing away from the light source are dark (directional light). You can even observe the directional light on the plane if you rotate in a circle with the A/D keys (more local illumination as well).

Q1: Was this lab fun? 1-10 (1 least fun, 10 most fun)
Q2: How was the write-up for the lab? Too much hand holding? Too thorough? Too vague? Just right?
Q3: How long did this lab take you?
Q4: Any other comments?

To submit this lab, zip together your source code and README.txt with questions. Name the zip file <HeroName>_L05.zip. Upload this on to Canvas under the L05 section.

LAB IS DUE BY **FRIDAY OCTOBER 07 2022 11:59 PM**!!