Attached is the Spring 2015 framework textures changes. To use this archive, download it and extract it in the directory containing your assignment 1 files when you decide to move onto assignment 2. Assignment 2 buillds off of your work from assignment 1. This archive does NOT include any of the files from assignment 1. It just provides you with some new code for dealing with textures, as well as an example shader for demonstrating multitexturing.

Notice that we are not providing any code for how to use these new Texture and TextureManager shader classes, nor how to call the sample_texture shader program. The archive does contain an updated sample.exe which demonstrates this shader in action, with texture support. It does not demonstrate anything else for assignment 2. Instead, we have provided a folder full of images and a sample video demonstrating a bunch of aspects of the assignment. It is by no means exhaustive. You are expected to figure out how much of the rest of the assignment should look.

The archive includes the texture files (two TGAs, one for diffuse and one for specular) you are expected to use in your assignment.

As always, if you have any questions or concerns with the assignment, the TAs (if assigned) and I are here to help. We recommend you start on this assignment as soon as you can, as it may take longer than the last assignment.

Thank you.

Frequency Asked Questions -- Assignment 2

Q: If I am using the framework, do I need to do anymore work with loading shaders, having correct error handling, or anything else in the first part of the assignment?
A: No, not really. The framework provides a ShaderManager class which takes care of this for you. It is to your advantage to read and understand how this class works, however.

Q: How do I use the ShaderManager or add shaders to it?
A: You need 1 shader manager to do the entire assignment. The manager's job is to store different shaders which are each identified by their own ShaderType enum value. For instance, you could add another value in the enum called PHONG, then you would register your shader something as follows:

```
shaderManager->RegisterShader(ShaderType::PHONG, "phong.vert",
"phong.frag")->Build();
```

Notice the importance of the "->Build()" part at the end. It is super important to remember to build the shader, otherwise it won't compile. After building the shader (which should only be done on application start-up), you can then use it to draw stuff by binding it:

```
std::shared_ptr<ShaderProgram> program = shaderManager-
>GetShader(ShaderType::PHONG);
program->Bind();
// pass uniforms and render objects
program->Unbind();
```

That's all there is to it. Just by changing which enum value you pass to GetShader, you can leverage the same block of code across multiple shaders. This significantly reduces the C++ code you need for this assignment.

Q: How do I use the TextureManager?
A: Nearly the same as the ShaderManager. It was designed to closely mimic the ShaderManager to make it easier to understand and use. You again only need 1 instance of a TextureManager for your entire application. Each texture you load is identified by an enum value in the TextureType enum. For assignment 2, you will probably only use something like ROOF_DIFFUSE and ROOF_SPECULAR. Otherwise, you can load textures on startup in almost an identical way to shaders. **Don't forget to build the texture after registering it with the texture manager!** Binding textures is also very similar to binding shader programs. However, I compressed down the code needed to do this. The code you need to call both binds a texture and attaches it to a uniform in your shader. An example of this might be:

```
// shader program is bound
// some uniforms set in the shader program
textureManager->BindAttach(TextureType::ROOF_DIFFUSE, program,
"diffuseSampler");
// set more uniforms
// draw something
textureManager->Unbind(TextureType::ROOF_DIFFUSE); // DON'T FORGET THIS
PART
// unbind the program
```

And that runs every loop. You can easily extend it to use more textures, as well. Feel free to read into how the TextureManager handles multitexturing.

Q: How do I reload textures during runtime?
A: If you have some function which loads and builds all of your shaders when the application starts up, simply calling this again gives you direct support for reloading textures. This is actually why the texture manager was designed with associating textures to enum values in the first place. It allows one level of indirection for all code accessing textures (similar to handles), which allows the manager to replace the textures in the background without any front-end code knowing they were changed. This is a super convenient design pattern for graphics systems.

Q: How do I reload shaders during runtime?
A: Although not required for the assignment, you SHOULD ABSOLUTELY DO THIS. It saves a lot of development time for this assignment if you can just make a change in the shaders and immediately see them while your application is running just by clicking Reload Shaders. This is also semantically the same as reloading textures. Simply put all of your shader loading and building code into 1 function and call it again by the press of a key or ImGui button. This will reload shaders and replace the existing ones with the new ones based on the enum values.

Q: How can I tell if my shaders are correct?
A: There are some samples provided to help you see what phong shading and blinn-phong shading look like, as well as what the animating lights look like. It is left up to you to try and figure out if your phong lighting looks right or not. Hint: the quality of phong lighting depends on how close you are to the model and how many triangles are in it. Nevertheless, both Tyler and I are willing to look at your output and see if it looks right. We cannot easily verify your code, so we prefer you don't ask us to unless you are dealing with a bug. In graphics, it is usually significantly easier to verify correctness visually.

We will be adding more Q&As to this list if more come in. We also synthesized a couple of extra questions and answers to help you with some things that you probably should be doing, but might not be (such as reloading shaders). That being said, please do let us know if you have any additional questions or seek clarification on any answers provided above.

Thanks!