Project 1: Intro to Graphics Engineering

Publish | Edit | ⋮

GPR-300 Intermediate Graphics & Animation Programming
Instructor: Daniel S. Buckstein
Project 1: Intro to Graphics Engineering

Summary:
In this project we are introduced to the course framework and delve into writing our own renderer code from scratch. We will explore and implement a variety of basic lighting and shading algorithms using OpenGL and the OpenGL Shading Language (GLSL), in a starter framework.

Objectives:
Upon successful completion of this assignment, you will have accomplished the following:

- Set up the course requirements.
- Set up an introductory OpenGL rendering pipeline in C/C++ and OpenGL.
- Set up an intermediate OpenGL rendering pipeline using the OpenGL and C-based course framework, *animal3D*.
- Implement a variety of lighting shaders.

Submission:
Start your work immediately by ensuring your coursework repository is set up, and public. Create a new main branch for this assignment. ***Please work in pairs (see team sign-up). Begin your submission immediately, copy the following into the text box, decide on your repository branch name for this assignment and fill in the information below.*** **The submission box locks at the specified deadline; be proactive and don't miss it. Late submissions, even by a minute, will not be accepted.**

Copy, edit and submit the following text once as a team (you do not need the

headings, please just provide your info as shown in the examples here):

1. ***Names of contributors***: Write the names of the contributors of this assignment.
   e.g. **Dan Buckstein**
2. ***A link to your public repository online***: Grab the clone link from your working repository, which should end with "*.git*".
   e.g. **https://github.com/dbucksteinccorg/graphics2-coursework.git** (note: not a real link)
3. ***The name of the branch that will hold the completed assignment***: Create a new branch for this project, submit only the name of this branch.
   e.g. **project0-main**
4. ***A link to the read-me and user instructions for this project***: Ensure your repository includes a read-me that summarizes how to use your finished product.
   e.g. **https://github.com/dbucksteinccorg/graphics2-coursework/blob/project0-main/project0-readme.pdf** (note: not a real link)
5. ***A link to your video (see below) that can be viewed in a web browser***: Ensure your video is public or shared with your instructor.
   e.g. YouTube link: **https://www.youtube.com/watch?v=OqOyxQVs8lY** (note: "Attack on Game Development" by Will Gordon & Connor Breen)

Finally, please submit a ***10-minute max*** demo video of your project. Use the screen and audio capture software of your choice, e.g. Google Meet, to capture a demo of your project as if it were in-class. This should include at least the following, in enough detail to give a thorough idea of what you have created (hint: this is something you could potentially send to an employer so definitely show off your professionalism and don't minimize it):

- Show the final result of the project and any features implemented, with a voice over explaining what the user is doing.
- Show and explain any relevant contributions implemented in code and explain their purpose in the context of the assignment and course.
- Show and explain any systems source code implemented, i.e. in framework or application, and explain the purpose of the systems; this includes changes to existing source.
- **DO NOT AIM FOR PERFECTION, JUST GET THE POINT ACROSS**. Please mind the assignment rubric to make sure you have demonstrated enough to cover each category.
- **Please submit a link to a video visible in a web browser, e.g. YouTube or Google Drive.**
- **Instead of waiting until last-minute for the video to upload, create a folder or links document on Drive that is accessible to your instructor. Submit the link to this folder as part of your official submission, and copy your video file/link there when it is ready.**

Setup Tutorial:

The following steps must be taken to prepare for this assignment and the rest of the course (repeat steps on any computer used for development throughout the semester). ***Read the following instructions carefully, and don't rush through them***:

1. **Virtual machines**: In this step, we set up our virtual machines, if needed. ***NOTE: This step is useful if you are learning remotely and do not have a graphics-capable machine. If you have a decent modern graphics card, skip to step 2***.
   A. Go to Pineapple and login using your Champlain credentials.
   B. Follow this tutorial to set up VMware and access your virtual machine instances.
      - Upon successful setup, you should open up VMware Horizon. Your view of the software may have virtual machines (VMs) with different names (e.g. I have faculty because I am faculty, yours will probably say student).
      - The VMs you may see are:
        - **GRID**: This is a ***persistent*** image that you log into as if you were on campus. Using a GRID machine is the same as logging into a physical computer in one of our labs, except it is virtual. It has high-powered graphics capabilities, all of the game development software required (namely Visual Studio) and low latency.
        - **Remote Student VDI**: This is a ***non-persistent*** image that has been stripped of development capabilities. Anything you create while logged into one of these instances is deleted when you log out.
          - ***This image is NOT to be used for development, or any task involving graphics and/or rendering***.
2. **Supporting software**: In this step, we set up our supporting software that will be useful throughout the course.
   A. You may use your personal machine or an instance of your GRID VM (signed in using your Champlain credentials). Again, GRID will be useful if your personal computer does not have capable GPU technology.
   B. You must have some sort of Git client installed. If you want a nice GUI, I recommend **SmartGit** **(https://www.syntevo.com/smartgit/)** .
      - Download SmartGit, extract and run the installer; launch when finished.
      - When prompted to select your license, select "***Non-commercial use only***".
      - The user agreement on the next page has a 10-second countdown; wait it out, check the boxes and click OK.
      - Setup your default user information:
        - If you have a GitHub account, input your username and GitHub-provided address (<username>@users.noreply.github.com).
        - If you are using Pineapple, add your username and Champlain email.
        - If you are using some other repository host, input your information accordingly.
      - Use SmartGit as SSH client.
      - Choose your default layout:
        - *Commits (Log History)*: presents a commit graph, which is more visually appealing if you like to see your commits and branches over time.

- *Working Tree (default)*: presents the file structure of your repository.
- Select privacy settings.

C. Visual Studio: Ensure you have Visual Studio 2019 installed on your machine.
- GRID VM has Visual Studio 2019 Enterprise already installed.
- On your personal machine, you may install Visual Studio 2019 Community with a Microsoft account (use non-Champlain email).
- On first launch, configure Visual Studio such that you are using the Visual C++ development environment (***NOT the default, so don't just click OK***).

D. GLSL Language Integration: Ensure Visual Studio is closed and install this plugin to highlight GLSL syntax in the text editor (VS2019): **https://marketplace.visualstudio.com/items?itemName=DanielScherzer.GLSL (https://marketplace.visualstudio.com/items?itemName=DanielScherzer.GLSL)**

E. ***Do not set up any supporting software on the non-persistent image because it will be deleted.***

3. **Course frameworks**: Here we set up the codebases used throughout the course.

A. ***Clone developer SDKs***: First, we set up the DevSDKs, a repository that contains some reusable headers and libraries.
- Using the Git interface of your choice, clone this repository: **https://github.com/dbucksteinccorg/DevSDKs-GPRO-2020.git (https://github.com/dbucksteinccorg/DevSDKs-GPRO-2020.git)**
- Check out the branch "*dev-sdks*".
- Navigate to the repository root (wherever you cloned it) in Windows Explorer.
- From the root, run the file "*DevSDKs/dev_install.bat*" ***as an administrator***.

B. ***Create and clone your working repository***: Next, we set up your repository for completing course work and submitting assignments.
- Using the Git hosting platform of your choice (e.g. GitHub, Pineapple, Bitbucket), create a new Git repository. This will be the repository you use to complete your assignments; you will submit the link to the repository with each assignment.
  - ***The repository must be completely empty***: no read-me, no license, no ignore; these will be filled in during the next part.
- Using the Git interface of your choice (e.g. command line, Git bash, SmartGit, TortoiseGit), clone the repository to your GRID instance. This can be done using the "*git clone*" command or some menu option in a GUI. Upon successful cloning, you will have a remote called "*origin*" and the empty repository on your machine.
- Since you own this repository, you must ensure you can push to it; some Git interfaces require you to "set upstream" to your remote ("*origin*") before you can push to it.
  - Using command line, this is done when you push a branch as follows: "*git push -u origin <local branch name>*"
- ***If you are working with a teammate, you only need one repository as a pair. Ensure both teammates have write access to the repository.***
- ***If you are using GitHub, the repository should be public.***

- ***If you are using Pineapple, you must ensure your instructor has access as faculty.***

C. ***Pull course framework, "animal3D"***: Next, we set up the repository from which you will receive starter code for lessons and projects.
   - After cloning your working repository, copy the course repository link: **https://github.com/dbucksteincc/animal3D-SDK-202101SP-Graphics2.git (https://github.com/dbucksteincc/animal3D-SDK-202101SP-Graphics2.git)**
   - In your Git interface, add a remote to the repository cloned in part B. "Remote" simply refers to another version of the repository hosted online; the default alias for a remote is "*origin*" (used to reference your working repository), so this one can have an alias like "*course*" to indicate that it is not yours. The command to add a remote is "*git remote add <alias> <url>*" or find this command in your GUI.
   - To pull content from this remote, you must first check if there are new branches; this action is called "fetch". Once new branches have been fetched, you can pull their latest commits.
   - In the course repository, there are 3 branches to start: "*main*" (formerly known as "*master*", recently changed industry-wide for political correctness), "*dev*" and "*dev-graphics2*"; the latter is where things will ultimately be made integrated and extended upon for this course.
   - Checkout the "*dev-graphics2*" branch.
   - ***The course repository is read-only and will be updated frequently. Fetch new branches at the start of every class.***

D. ***Push repo content to your working remote***: Finally, we copy the cloned content from the course repo to your working repo.
   - Make sure you have "set upstream" of your local repository so that it points to "*origin*" (your working repo).
   - Push all branches.
   - Looking at your working repository online, you will see that it now contains the content provided from the course repo!
   - ***This step will be repeated frequently throughout the course. Commit and push often, and don't forget to push your work when submitting assignments!***

E. ***Do not set up any repositories on the non-persistent image because they will be deleted.***

4. **Configuring the framework**: Here we explore the framework and its debugging features.

A. Upon checking out the "*dev-graphics2*" branch, you will see that your repository root contains a folder called "*animal3D SDK*" (henceforth referred to as the "***framework***"). The framework is structured as follows:
   - *./include/* - This is where framework headers live.
   - *./project/* - This is where the Visual Studio projects and solutions live.

- *./resource/* - This is where resource and configuration files live, including shaders, models and textures.
- *./source/* - This is where your C/C++ source and header files should live.
- *./utility/* - This is where internal utilities live.
- *./CONFIG_VS_RUN-AS-ADMIN_git.bat* - This will launch the Visual Studio solution intended to configure your Visual Studio environment; it must be used once per machine before using the actual framework.
- *./LAUNCH_VS.bat* - This will launch the Visual Studio solution intended to be run and debugged locally; for now, the solution consists of two projects: the "plugin" and the "player".
- *./PACKAGE_VS.bat* - This is a largely unused utility to collect all source and resources into a folder.

B. Configuring the framework:
- Run the config script ("*CONFIG_VS_RUN-AS-ADMIN_git.bat*") ***as an administrator***, which first opens a console window, then opens a Visual Studio solution called "*animal3D-VSSetPath*" containing a configuration project. ***DO NOT CLOSE THE CONSOLE***.
- Build with F7 or Ctrl+Shift+B. When configuration succeeds, it will say so in the build output. There is no application to run, so trying to do so will throw an error.
- Close Visual Studio and ***FOLLOW THE PROMPTS IN THE CONSOLE*** (it will clean up after itself).

C. Launching and using the framework:
- Run the launch script ("*LAUNCH_VS.bat*") ***as an administrator***, which opens a Visual Studio solution called "*animal3D-sdk*" containing two projects:
  - "*animal3D-DemoPlayerApp*": This is the application itself, which produces a "player" window into which a "demo plugin" can be loaded and played.
  - "*animal3D-DemoPlugin*": This is the debuggable demo plugin for development. ***Your work happens here***.
- Upon building the solution with F7 or Ctrl+Shift+B and then running it with F5, you will get an empty window with a File menu. Here is a brief description of how to get drawing in the player window:
  - There are 3 options for loading the debuggable plugin:
    I. *Load as-is*: "**File > DEBUG: Demo project hot build and load > Load without debugging**": If you have recently made updates to the plugin project and built in VS, this is the fastest option. It will load the plugin as it was most recently compiled.

    II. *Hot-build and load*: "**File > DEBUG: Demo project hot build and load > Quick build and load**": Any files changed in the plugin project since the last build will be compiled in real-time, while the demo runs. The demo will continue to play while the build status is printed in the console. During this time, you will lose the ability to debug. The advantage to this feature is that

you do not need to close the player to build; it is a live "code injection" much like Unity or Unreal.

III. *Hot-rebuild and load*: "**File > DEBUG: Demo project hot build and load > Full rebuild and load**": In a similar fashion as above, the plugin project will rebuild all source files instead of just those changed since the last build.

- In addition to the debuggable plugin, you may also load "pre-built" plugins; these will be useful for things like double-checking your work against the instructor's completed version of assignments. Go to "**File > Load demo...**" and select the demo from the list.

5. **Please mind the following at all times**:
   - ***NEVER open either solution on its own, i.e. by opening Visual Studio first or navigating to the solution file.***
   - ***ALWAYS use the "LAUNCHVS" batch scripts provided because they configure things correctly (feel free to read them).***
     - ***Failure to use the launcher utilities will prevent you from building your projects!***
   - ***If using the GRID VM, when you are finished developing, shut down the GRID machine instance. All progress will be retained.***

Instructions & Requirements:

DO NOT begin programming until you have read through the complete instructions, bonus opportunities and standards, start to finish. Take notes and identify questions during this time. The only exception to this is whatever we do in class.

Using the provided framework, implement the following:

1. **OpenGL tutorial**: Visit the following pages and complete the tutorial described below.
   A. **The OpenGL "Blue Book"** (https://f.usemind.org/files/b/1/UseMind.ORG_comprehensive-tutorial-and-reference_2015__.pdf) : The OpenGL SuperBible, colloquially referred to as the "***Blue Book***" is a comprehensive official reference for OpenGL and GLSL programming.
   B. **The OpenGL "Red Book"** (https://www.cs.utexas.edu/users/fussell/courses/cs354/handouts/Addison.Wesley.OpenGl : The OpenGL Programming Guide, colloquially referred to as the "***Red Book***" is the official guide to learning OpenGL. Throughout the course we will discuss elements of OpenGL in the context of the provided renderer.
   C. ***Intro to OpenGL programming***: Complete Chapter 2 of the **Blue Book** (https://f.usemind.org/files/b/1/UseMind.ORG_comprehensive-tutorial-and-reference_2015__.pdf) (pages 53 - 67) using the course framework, *animal3D*. The code in this chapter is in C++ with classes; here you should do it in C using only the functions provided in animal3D. This section walks you through the direct implementation of the sample code in animal3D. We will do some work on this part in class.

- *Tutorial branch*: The repository contains a starter branch called "*graphics2/tutorial1a*"; check out this branch before proceeding.
- *Rendering*: For the purposes of this exercise, the animal3D equivalent of code in Listings 2.1 and 2.2, the function called "*render*", can be found in the plugin project as follows:
  - Expand the "*animal3D-DemoPlugin*" project.
  - Navigate to the "*Source Files/common/A3_DEMO/a3_DemoState*" filter and open the C source file "*a3_DemoState-idle-render.c*".
  - All rendering code for this tutorial may be completed in the function "*a3demo_renderTest*", which is invoked in the function "*a3demo_render*".
  - Clarification: you are ***not*** implementing the sample class itself, only what is in the function.
  - You will see that this file contains more stuff that we will explore later.
- *Startup, setup*: The animal3D equivalent of code Listings 2.5 and 2.6, the functions called "*compile_shaders*" and "*startup*" can be found in the plugin project as follows:
  - In the same filter as the above, open the C source file "*a3_DemoState-load.c*".
  - All setup code for this tutorial may be completed in the function "*a3demo_loadShaders*". For the purposes of this tutorial, you may implement the above sample code at the end of this function for simplicity.
  - You will see that this file contains more stuff that we will explore later.
- *Shutdown, cleanup*: The animal3D equivalent of "*shutdown*" in the above listings can be found as follows:
  - In the same filter as the above, open the C source file "*a3_DemoState-unload.c*".
  - All cleanup code for this tutorial may be completed in the function "*a3demo_unloadShaders*".
  - You will see that this file contains more stuff that we will explore later.
- *Data storage*: The member variables of the sample class, the ***graphics handles***, must be persistent in memory. For the purposes of this assignment, the animal3D equivalent of the class which stores the data can be found as followed:
  - Navigate to the "*Header Files/A3_DEMO*" filter and open the header file "*a3_DemoState.h*".
  - The data structure that should hold the handles is called "*a3_DemoState*" as you will see there is a persistent instance of it passed to all of the above functions (in the same fashion as the "*this*" pointer would be passed for the sample class in C++). Add the data members to the end of the data structure.
  - You will see that this data structure contains more stuff that we will explore later.

2. **Enter graphics engineering**: The goal of this project is to explore OpenGL both on its own and using a rendering framework as your first foray into graphics engineering. The course framework is set up to get you going with the following primary tasks:

A. ***Project branch***: Check out the project starter branch "*graphics2/proj1*" to begin the project. Final friendly reminder to create your new branch before beginning any programming tasks. Do not work on the "*dev-graphics2*" branch.
B. ***Explore renderer utilities***: Now that you have been exposed to the basics and fundamentals of the requirements of a renderer, it is time to explore the provided rendering framework: ***animal3D***. The framework has a variety of utilities that encapsulate the core functionalities of an OpenGL renderer. It is largely responsible for efficient organization and setup of things like shaders and geometry, the start of which you completed in part 1. This section walks you through a more advanced render pipeline using the animal3D utilities and the tasks you should do to get acquainted with the framework. ***All programming takes place in the "animal3D-DemoPlugin" project, where you will repeatedly visit and modify these files. Please review the following files BEFORE changing any code***, then follow the instructions below:
    I. *Demo state*: All persistent renderer data for the active plugin is stored in a data structure referred to as the ***demo state***. Within the plugin project, the declaration of this data structure is found in "*Header Files/A3_DEMO/a3_DemoState.h*" (moving forward, referred to as ***demo state [header/interface]***). We saw a simplified version of this in the tutorial, but now it contains data structures that encapsulate graphics elements.
    II. *Demo callbacks*: The ***callbacks*** are the connections between the plugin and the player window. When an event happens in the window (e.g. mouse click), the window calls the appropriate plugin function to interface with your stuff. The callbacks source is in "*Source Files/common/A3_DEMO/_a3_demo_callbacks.c*" (referred to as ***callbacks [source/implementation]***); review the file to see what types of events are handled and how.
    III. *Demo loading and unloading*: When the plugin is loaded and unloaded, some of the callbacks result in loading and unloading tasks, such as setting up graphics. These source files are under "*Source Files/common/A3_DEMO/a3_DemoState/*" and are called "*a3_DemoState-load.c*" (***loading***) and "*a3_DemoState-unload.c*" (***unloading***).
    IV. *Demo idle loop (input, update, render)*: The ***idle loop*** is where all the real-time action happens. Also known as the "game loop" (used more generally here), the idle loop consists of three main tasks: handling input, updating the demo state and rendering the scene. These events are invoked through the "idle" callback (called whenever the window has no other messages to process) and can be

    found in "*a3_DemoState-idle-input.c*" (***input***), "*a3_DemoState-idle-update.c*" (***update***) and "*a3_DemoState-idle-render.c*" (***render***).
    V. *Demo shader program utility*: Shader programs are managed by the framework, but on the demo front-end there exists a small interface for specifying how shaders are used in this specific build. The structure affiliates a set of uniform

handles with a program for convenience. Find the data structure in "*Header Files/A3_DEMO/_a3_demo_utilities/a3_DemoShaderProgram.h*" (***shader***); this will be changed often.

VI. *Demo modes*: The demo modes represent individual modes or scenes as part of the overall demo. The first one is called "*a3_DemoMode0_Intro*" and has an associated header and source file set. We will see a bunch of these.

C. ***Implement renderer utilities***: Now that you have familiarized yourself with the common features of the framework, complete the following programming tasks in the specified files (***following the ****TO-DO prompts, looking around for hints and building frequently to test***) to gain a better understanding of how *animal3D* actually works and manages itself. As a graphics programmer you will be responsible for navigating custom tech and manipulating it to fulfill a variety of tasks. Here are the steps for this demo:

I. *Explore pre-built example*: Load the pre-built example demo for this project: "*File > Load demo > ...Proj1*". The demo shows the completed scene with procedural and loaded models, the skybox and the required shader effects.

II. *Initial build and run*: Either build the project in Visual Studio then launch using "*File > DEBUG... > Load without building*", or use "*Quick build*" to hot-build and run the demo directly through the player window. The demo is now entirely broken and you will see a pitch-black screen.

III. *Setup 3D models for rendering*: Complete the following steps to prepare the scene's geometry; no point in any of this without geometry.

- First, open the ***demo state header*** and uncomment the three blocks of code affiliated with models: ***draw data buffers***, ***vertex arrays*** and ***drawables***. The purpose of this step is to show you where they are and what they mean, so pay attention:
  - The ***buffer*** interface maps your application data (in this case, geometry, but it can include other things... later!) with the GPU buffers used for drawing. The interface encapsulates a standard graphics feature. In OpenGL documentation, please read about ***buffers***, namely ***vertex buffer objects (VBO)***.
  - The ***vertex array*** interface describes the buffer data to the shaders and helps with rendering calls. This also encapsulates a standard graphics feature. In OpenGL documentation, please refer to ***vertex array objects (VAO)***. Pro tip: you created a blank descriptor in the tutorial in part 1!
  - The ***drawable*** interface associates the above two features and simply describe the model you want to draw. This is ***not*** a standard graphics feature and was created for convenience.
- Next, open the ***loading source*** and go to the "*a3demo_loadGeometry*" function. This is where the models are prepared.
  - Complete the TO-DO's using data in the demo state and this function; use the provided hints to accelerate the process.

- Finally, open the ***unloading source*** and uncomment things in the "*a3demo_unloadGeometry*" function.

IV. *Setup shader programs*: Complete the following steps to prepare the GLSL shaders and programs.
   - First, in the demo state, uncomment the ***shader programs***.
     - ***Shader programs*** are a standard graphics feature that describe some drawing pipeline; it is made up of individual ***shaders***.
     - The ***shaders*** themselves are the building blocks of the programs and do not need to be persistent, which is why they are not stored in the demo state. We will discuss a variety of shaders throughout.
   - Next, open the loading source, uncomment and implement things in the "*a3demo_loadShaders*" function.
     - At this point, hold off on the uniforms at the bottom of the function, they are explained below.
   - Finally, uncomment things in the "*a3demo_unloadShaders*" function in the unload source.

V. *Setup textures*: Complete the following steps to prepare the textures.
   - First, in the demo state, uncomment the ***textures***.
     - ***Textures*** are a type of organized data buffers on the GPU typically used to represent images; they have other uses too!
   - Next, open the loading source, uncomment things in the "*a3demo_loadTextures*" function.
   - Finally, uncomment things in the "*a3demo_unloadTextures*" function in the unload source.

VI. *Additional setup*: So far, you can see a pattern being established when it comes to managing your graphics objects: adding to demo state, loading and setup, unloading and cleanup. There are a few additional steps including, most importantly, the consumption and usage of these objects for rendering. Complete the following steps to make sure the framework is all set:
   - In loading, uncomment the contents of the "*a3demo_loadValidate*" and "*a3demo_initDummyDrawable_internal*" functions. These functions ensures that everything will work correctly if you use the hot-build feature.
   - In unloading, uncomment the contents of the "*a3demo_unloadValidate*" function. This function reports whether any graphics objects have not been properly released.
   - Navigate to the intro demo mode filter: "*Source Files/common/A3_DEMO/a3_DemoMode0_Intro*"; open the update source for this mode: "*a3_DemoMode0_Intro-idle-update.c*". Here you must update the scene objects so that they display correctly.
   - Navigate to the render source for this mode: "*a3_DemoMode0_Intro-idle-render.c*". Here you must uncomment the existing rendering pipeline and, later, implement a couple more things.

3. **Implement shaders**: Using the pipeline implemented in part 2:
   A. ***Encode/decode shaders***: The encoded shaders are enabled to help ensure the framework is functional. You will see in the function "*a3demo_loadShaders*" in file "*a3_DemoState-load.c*" that the shaders you need for this project are already decoded. This is because, for this part, you need to write and use your own shaders. For testing purposes, add the "*e/*" sub-directory to the path (make sure you double-check in Windows explorer that it is correct). For your actual implementation, remove the "*e/*" from each shader file path.
      - With part 2 complete and encoded shaders enabled, your runtime should match the pre-built demo.
        - The only thing left to uncomment is the uniform configuration in "*a3demo_loadShaders*". These must be uncommented to be able to send uniform data to shader programs. Uncomment this part now if you haven't already.
      - With encoded shaders ***disabled*** (required for you), everything is broken!!!
   B. ***Implement shader programs***: Implement the following effects by visiting and completing the following GLSL files (navigate to "*Resource Files/A3_DEMO/glsl/4x*"):
      I. *Solid color*: This shader program renders the affected models with a solid color.
         - Code setup: In "*a3_DemoMode0-idle-render.c*", make sure all of the required code is uncommented. When it comes time to draw objects (switch statement at line 249), follow the to-do prompts under "*case intro_renderModeSolid*" to pass a matrix and color to the program. Use the surrounding code for hints.
         - Vertex shader: Open "*vs/passthru_transform_vs4x.glsl*" and follow the to-do prompts.
         - Fragment shader: Open "*fs/drawColorUnif_fs4x.glsl*" and follow the to-do prompts.
      II. *Texturing*: This shader program renders the affected models with a texture blended with a solid color.
         - Code setup: In the render function, follow the to-do prompts under "*case intro_renderModeTexture*" to activate the appropriate diffuse texture for each model. Use the surrounding code for hints.
         - Vertex shader: Open "*vs/00-common/passTexcoord_transform_vs4x.glsl*" and follow the to-do prompts.
         - Fragment shader: Open "*fs/00-common/drawTexture_fs4x.glsl*" and follow the to-do prompts.

      III. *Lambert shading*: This shader program renders the affected models using the ***Lambertian shading model***. This shading model uses the dot product of a ***unit surface normal*** (a vertex attribute) and a ***unit light direction*** (vector from surface to light) to approximate how much light hits a surface at any given point. See Blue Book for examples.

- Code setup: First, in the shader header, you will need to add a few uniform handles to describe light parameters: the positions of a set of lights in view space, the colors of the lights, and the radii of the lights. These uniforms must be configured along with the others in "*a3demo_loadShaders*". In the render function, before the objects are rendered, there is a to-do prompt to send the lights' data to the active program. Finally, follow the to-do prompts under "*case intro_renderModeLambert*" to send the correct matrices for each model. Use the surrounding code for hints.
- Vertex shader: Open "*vs/00-common/passTangentBasis_transform_vs4x.glsl*" and follow the to-do prompts.
- Fragment shader: Open "*fs/00-common/drawLambert_fs4x.glsl*" and follow the to-do prompts.

IV. *Phong shading*: The ***Phong shading model*** extends Lambert by incorporating a specular highlight. This is the dot product of the ***unit reflected light vector*** (calculated using normal and light vector) and the ***unit view vector*** (vector from surface to eye).
- Code setup: In the render function, follow the to-do prompts under "*case intro_renderModePhong*" to activate the appropriate specular texture for each model. Use the surrounding code for hints.
- Vertex shader: This program uses the same shader as above, "*vs/00-common/passTangentBasis_transform_vs4x.glsl*" and does not require any changes. Since shaders are decoupled from programs, they can be used multiple times to facilitate multiple programs.
- Fragment shader: Open "*fs/00-common/drawPhong_fs4x.glsl*" and follow the to-do prompts. Optionally, you may use "*fs/00-common/utilCommon_fs4x.glsl*" to write functions that would be common to multiple lighting algorithms to avoid redundancy; this file is appended to the end of the Lambert and Phong shader files. Simply declare any functions to be used in these files above main.

4. **Testing & demonstration**: You must thoroughly test your render pipeline and shaders. Your demonstration must show evidence of the following:
   A. **Walkthrough and justification of architecture**: Demonstrate that the above requirements have been met in code.
   B. **Starter tutorial**: Demonstrate completion of the introductory tutorial in chapter 2 of the Blue Book.
   C. **Framework**: Demonstrate completion of the intermediate tutorial using animal3D data structures and functions.
   D. **Shaders**: Demonstrate completion of the shaders implemented throughout the project (solid color, texturing, Lambert, Phong). Furthermore, you must demonstrate that the shaders are your own and not the encoded files.
   E. **Takeaways**: Discuss personal and professional takeaways from this project.
5. **Pro tips**: Mind the following pro tips to help you avoid headaches, both now and in the

future:

A. ***Source file extensions****:* You can switch between the C and C++ programming languages simply by changing the extension of a source file: "*.c*" is for C, and "*.cpp*" is for C++. The C programming language is mostly compatible with C++, but not the other way around. The course framework is and always will be written in a way that is compatible with both C and C++ whenever possible.
B. ***Points of interest in framework***: The framework may seem complicated at first, but it is clearly labelled with steps that will help you understand its architecture and the tasks at hand. Look for the ****TO-DO** tag throughout the code to find places where something must be done for the assignment.
C. ***You must read and do independent research***: There are a vast amount of resources presented in the course materials, and this briefing alone, to help you complete the assignment; you ***must not expect*** that every answer will be provided outright. You will be challenged throughout the course to inquire and infer, research and develop, and gain a solid experiential understanding of graphics engineering with the tools provided. Frequently refer to the Blue Book and the GLSL specifications for reference on OpenGL/GLSL functions, requirements and constraints. That said, your primary resources for the course framework are 1) your instructor, and 2) the discussion board on Canvas.
D. ***Use the encoded shaders for testing, but not submission***: The framework has several examples, including binary text files that encode shaders for testing. The engineering side is complex enough as-is, without bringing shaders into the picture, so you should continue to first test the framework with the encoded shaders, and then remove the encoding to implement your own. To remove encoding on a shader, simply remove the "*e/*" in the shader file path; shaders are loaded in "*a3_DemoState-load.c*" function "*a3demo_loadShaders*".
E. ***It's just data***: OpenGL and GLSL do not know what you are feeding it; they will only do what they're told, so you must constantly be aware of your data and how it is described. This assignment provides your first exposure to data management. Follow all instructions carefully, including all prompts in the provided code, for maximum success.

Bonus:
You are encouraged to complete one or more of the following bonus opportunities (rewards listed):

- ***Per-vertex vs per-fragment shading (+2)***: In your Phong shader program, demonstrate the difference between ***per-vertex*** and ***per-fragment*** shading. The former should appear blocky with visible geometry edges, whereas latter should be smooth with minimal visibility of geometry edges.
- ***Transform uniform buffer (+2)***: In the demo state, set up a ***uniform buffer*** for the

transform stack and lighting data. Use the buffer to fulfill the requirements of your shaders.

Coding Standards:
You are required to mind the following standards (penalties listed):

- ***Reminder: You may be referencing others' ideas and borrowing their code. Credit sources and provide a links wherever code is borrowed, and credit your instructor for the starter framework, even if it is adapted, modified or reworked (failure to cite sources and claiming source materials as one's own results in an instant final grade of F in the course)***. Recall that borrowed material, even when cited, is not your own and will not be counted for grades; therefore you must ensure that your assignment includes some of your own contributions and substantial modification from what is provided. ***This principle applies to all evaluations***.
- ***Reminder: You must use version control consistently (zero on organization)***. Commit after a small change set (e.g. completing a section in the book) and push to your repository once in a while. Use branches to separate features (e.g. a chapter in the book), merging back to the parent branch (dev) when you stabilize something.
- ***Visual programming interfaces (e.g. Blueprint) are forbidden (zero on assignment)***. The programming languages allowed are: C/C++, C# (Unity) and/or Python (Maya).
  - If you are using Unity, all front-end code must be implemented in C# (i.e. without the use of additional editors). You may implement and use your own C/C++ back-end plugin. The editor may be used strictly for UI (not the required algorithms).
  - If you are using Unreal, all code must be implemented directly in C/C++ (i.e. without Blueprint). Blueprints may be used strictly for UI (not the required algorithms).
  - If you are using Maya, all code must be implemented in Python. You may implement and use your own C/C++ back-end plugin. Editor tools may be used for UI.
  - You have been provided with a C-based framework called *animal3D* from your instructor.
  - You may find another C/C++ based framework to use. Ask before using.
- ***The 'auto' keyword and other language equivalents are forbidden (-1 per instance)***. Determine and use the proper variable type of all objects. Be explicit and understand what your data represents. Example:
  - auto someNumber = 1.0f;
    - This is a float, so the correct line should be: float someFloat = 1.0f;
  - auto someListThing = vector<int>();
    - You already know from the constructor that it is a vector of integers; name it as such: vector<int> someVecOfInts = vector<int>();
  - Pro tip: If you don't like the vector syntax, use your own typedefs. Here's one to make the previous example more convenient:
    - typedef vector<int> vecInt;

    - vecInt someVecOfInts = vecInt();
- ***The 'for-each' loop syntax is forbidden (-1 per instance)***. Replace 'for each' loops with traditional 'for' loops: the loops provided have this syntax:
    - In C++: for (<object> : <set>)...
    - In C#: foreach (<object> in <set>)...
- ***Compiler warnings are forbidden (-1 per instance)***. Your starter project has warnings treated as errors so you must fix them in order to complete a build. ***Do not disable this***. Fix any silly errors or warnings for a nice, clean build. They are generally pretty clear but if you are confused please ask for help. Also be sure to test your work and product before submitting to ensure no warnings/errors made it through. This also applies to C# projects.
- ***Every section/block of code must be commented (-1 per ambiguous section/block)***. Clearly state the intent, the 'why' behind each section/block. This is to demonstrate that you can relate what you are doing to the subject matter.
- ***Add author information to the top of each code file (-1 for each omission)***. If you have a license, include the boiler plate template (fill it in with your own info) and add a separate block with: 1) the name and purpose of the file; and 2) a list of contributors and what they did.
- ***Immediate mode is forbidden (zero on assignment)***: In this course we are studying modern graphics engineering principles; immediate mode refers to an ancient and deprecated set of OpenGL functions. The tutorials followed use the correct techniques; do not use tutorials on the internet that will lead you astray.

Points 10

Submitting a text entry box

Due	For	Available from	Until
-	Everyone	-	-

GraphicsAnimation-Master-Range-x2

Criteria	Ratings			Pts
IMPLEMENTATION: Architecture & Design Practical knowledge of C/C++/API/framework programming, engineering and architecture within the provided framework or engine.	**2 to >1.0 pts** **Full points** Strong evidence of efficient and functional C/C++/API/framework code implemented for this assignment; architecture, design and structure are largely both efficient and functional.	**1 to >0.0 pts** **Half points** Mild evidence of efficient and functional C/C++/API/framework code implemented for this assignment; architecture, design and structure are largely either efficient or functional.	**0 pts** **Zero points** Weak evidence of efficient and functional C/C++/API/framework code implemented for this assignment; architecture, design and structure are largely neither efficient nor functional.	2 pts
IMPLEMENTATION: Content & Material Practical knowledge of content relevant to the discipline and course (e.g. shaders and effects for graphics, animation algorithms and techniques, etc.).	**2 to >1.0 pts** **Full points** Strong evidence of efficient and functional course- and discipline-specific algorithms and techniques implemented for this assignment; discipline-relevant algorithms and techniques are largely both efficient and functional.	**1 to >0.0 pts** **Half points** Mild evidence of efficient and functional course- and discipline-specific algorithms and techniques implemented for this assignment; discipline-relevant algorithms and techniques are largely either efficient or functional.	**0 pts** **Zero points** Weak evidence of efficient and functional course- and discipline-specific algorithms and techniques implemented for this assignment; discipline-relevant algorithms and techniques are largely neither efficient nor functional.	2 pts
DEMONSTRATION: Presentation & Walkthrough Live presentation and walkthrough of code, implementation, contributions, etc.	**2 to >1.0 pts** **Full points** Strong evidence of accuracy and confidence in a live walkthrough of code discussing requirements and high-level contributions; walkthrough is largely both accurate and confident.	**1 to >0.0 pts** **Half points** Mild evidence of accuracy and confidence in a live walkthrough of code discussing requirements and high-level contributions; walkthrough is largely either accurate or confident.	**0 pts** **Zero points** Weak evidence of accuracy and confidence in a live walkthrough of code discussing requirements and high-level contributions; walkthrough is largely neither accurate nor confident.	2 pts
DEMONSTRATION: Product & Output Live showing and explanation of final working implementation, product and/or outputs.	**2 to >1.0 pts** **Full points** Strong evidence of correct and stable final product that runs as expected; end result is largely both correct and stable.	**1 to >0.0 pts** **Half points** Mild evidence of correct and stable final product that runs as expected; end result is largely either correct or stable.	**0 pts** **Zero points** Weak evidence of correct and stable final product that runs as expected; end result is largely neither correct nor stable.	2 pts

Criteria	Ratings			Pts
ORGANIZATION: Documentation & Management Overall developer communication practices, such as thorough documentation and use of version control.	**2 to >1.0 pts** **Full points** Strong evidence of thorough code documentation and commenting, and consistent organization and management with version control; project is largely both documented and organized.	**1 to >0.0 pts** **Half points** Mild evidence of thorough code documentation and commenting, and consistent organization and management with version control; project is largely either documented or organized.	**0 pts** **Zero points** Weak evidence of thorough code documentation and commenting, and consistent organization and management with version control; project is largely neither documented nor organized.	2 pts
BONUSES Bonus points may be awarded for extra credit contributions.	**0 pts** **Points awarded** If score is positive, points were awarded for extra credit contributions (see comments).		**0 pts** **Zero points**	0 pts
PENALTIES Penalty points may be deducted for coding standard violations.	**0 pts** **Points deducted** If score is negative, points were deducted for coding standard violations (see comments).		**0 pts** **Zero points**	0 pts
				Total Points: 10