



Assignment 01

The Canvas assignment is [here \(https://utah.instructure.com/courses/886852/assignments/12883235\)](https://utah.instructure.com/courses/886852/assignments/12883235).

Before trying to work on this assignment you should study and understand the following example programs:

- [Solution Setup \(https://utah.instructure.com/courses/886852/pages/solution-setup-example\)](https://utah.instructure.com/courses/886852/pages/solution-setup-example)
- [Static Library \(https://utah.instructure.com/courses/886852/pages/static-library-example\)](https://utah.instructure.com/courses/886852/pages/static-library-example)
- [Platform-Specific Code \(https://utah.instructure.com/courses/886852/pages/platform-specific-code-example\)](https://utah.instructure.com/courses/886852/pages/platform-specific-code-example)

The solution that you should start with to do this assignment can be downloaded [here \(https://utah.instructure.com/courses/886852/files/150334746/download\)](https://utah.instructure.com/courses/886852/files/150334746/download). 
(https://utah.instructure.com/courses/886852/files/150334746/download?download_frd=1) . The solution doesn't have a Graphics project, but the files that you will need to add to your new Graphics project that you create can be downloaded [here \(https://utah.instructure.com/courses/886852/files/150334773/download?wrap=1\)](https://utah.instructure.com/courses/886852/files/150334773/download?wrap=1). 
(https://utah.instructure.com/courses/886852/files/150334773/download?download_frd=1) .

If you want to make sure that the program will run on your computer when built properly (or you want an example of what your ZIP file should look like that others can download from your write-up page) you may download the reference implementation that I have made:

- [Direct3D reference implementation \(https://utah.instructure.com/courses/886852/files/150334765/download?wrap=1\)](https://utah.instructure.com/courses/886852/files/150334765/download?wrap=1). 
(https://utah.instructure.com/courses/886852/files/150334765/download?download_frd=1)
- [OpenGL reference implementation \(https://utah.instructure.com/courses/886852/files/150334763/download?wrap=1\)](https://utah.instructure.com/courses/886852/files/150334763/download?wrap=1). 
(https://utah.instructure.com/courses/886852/files/150334763/download?download_frd=1)

Requirements

- Rename the solution to match your GitLab project name
- Create a Graphics static library and add it to the solution
 - Set up the folder structure (on disk and in Solution Explorer) and the Visual Studio references
- Create a new game
- Configure your game
 - Change the caption window when the game runs
 - Change the icon
- Log a message when your game is initialized
 - Please include something specific about your game (either "MyGame" or the name of your game that shows up in the window title bar)
- Log a message when your game is cleaned up
 - Please include something specific about your game (either "MyGame" or the name of your game that shows up in the window title bar)
- Create a new fragment shader for your game that animates the triangle's color
 - Make sure that both platforms behave identically
- Your write-up should:
 - Show a screenshot of your game running
 - Make a screenshot that shows the triangle with a different color from the default white
 - (You may also show an animated GIF or video if you prefer)
 - Show a screenshot of the generated log file with your game's initialization and clean up messages
 - Tell us which projects needed to add a reference to the new Graphics project


- Did you find any projects whose code mentioned the Graphics namespace, but didn't need a reference added? If so, give an example, and explain why the code in question doesn't require a reference.
- Tell us about any thoughts you have about the engine code base that has been provided to you
 - How is it organized? Is there anything that you like or don't like about the organization?
 - What do you think of the code style? Is there anything that you like or don't like about it?
 - Is there anything that you were initially confused about? Tell us why you were confused and what you figured out to not be confused.
- Discuss briefly your expectations of the class based on the first lecture and what you know so far, and tell us what you hope to learn from it
 - Take some time and think about what you personally might be able to get out of the class that aligns with your interests

Submission Checklist

- Your write-up should follow the [standard guidelines for submitting assignments \(https://utah.instructure.com/courses/886852/pages/submitting-assignments\)](https://utah.instructure.com/courses/886852/pages/submitting-assignments) and the [standard guidelines for every write-up \(https://utah.instructure.com/courses/886852/pages/write-up-guidelines\)](https://utah.instructure.com/courses/886852/pages/write-up-guidelines)
- Your git repository should have no temporary files committed
 - Nothing listed in the .gitignore file should be checked into git
 - This includes any vcxproj.user files that set up the working directory for debugging. Do not commit them!
- If you delete the temp/ folder, build your game project *only*, and run the game:
 - A message should appear to the user saying that something has gone wrong
 - (If the game runs correctly and no message appears you have done something wrong with dependencies! Your game shouldn't be able to find any shaders and shouldn't be able to run without them.)
 - The game should exit **but not crash**
- If you delete the temp/ folder, build only your game and the BuildMyGameAssets projects, and run the game:
 - Everything should work correctly. This is a sign that your project dependencies are set up correctly, and is how the TA will correct your assignments.
 - Sometimes students get dependencies wrong and their projects fail when built the first time but then succeed the second time. Before submitting your finished assignment it is important to try building from scratch by deleting the temp/ folder to make sure that your dependencies are correct.
 - (Even if it works the first you may still have incorrect dependencies that wouldn't always work. If it doesn't work the first time, however, then you *know* that something is wrong and should fix it before submitting.)
- You should be able to run the game by double-clicking the executable in Windows Explorer
- Your downloadable game should be easy to download and run:
 - Does your write-up have a direct link to it? (Does clicking a single link allow the user to download it?)
 - Is it a ZIP file? (Not a RAR, not a 7ZIP, etc.)
 - Does your ZIP file have the LOG file from the last time *you* ran the game? It shouldn't!
 - Does your ZIP file have an extra root folder in it that the user has to click through? It shouldn't!

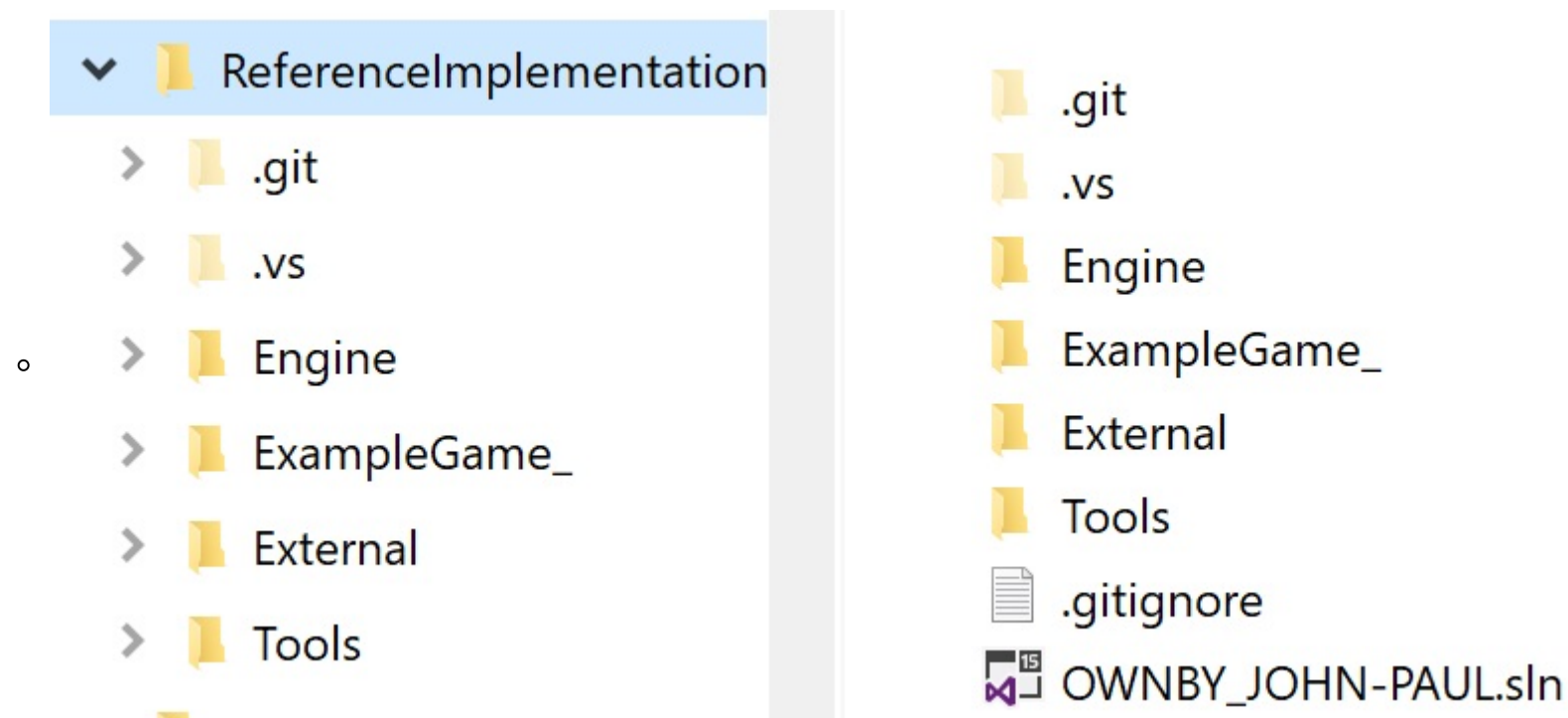
Finished Assignments

Details

You should start with the [solution provided by Tony \(https://utah.instructure.com/courses/886852/files/150334746/download\)](https://utah.instructure.com/courses/886852/files/150334746/download)  [\(https://utah.instructure.com/courses/886852/files/150334746/download?download_frd=1\)](https://utah.instructure.com/courses/886852/files/150334746/download?download_frd=1) for this assignment. You will only have to modify a few things for this assignment.

Solution Name

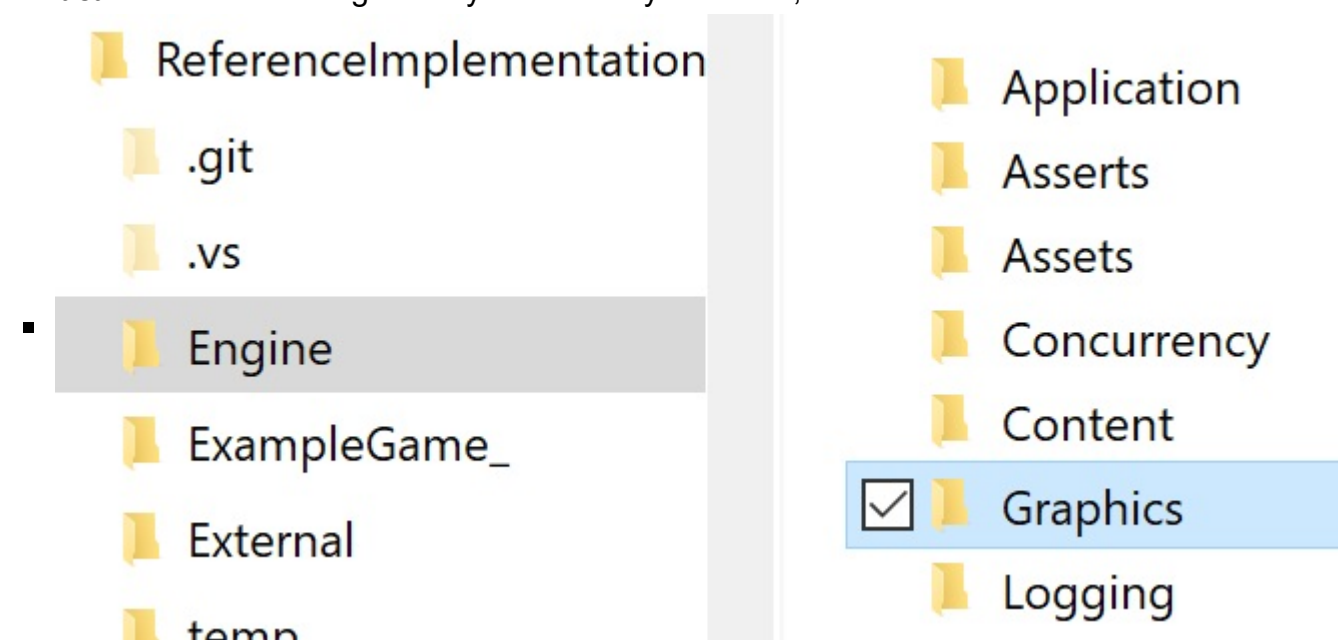
- Your solution's folder structure should look like this:



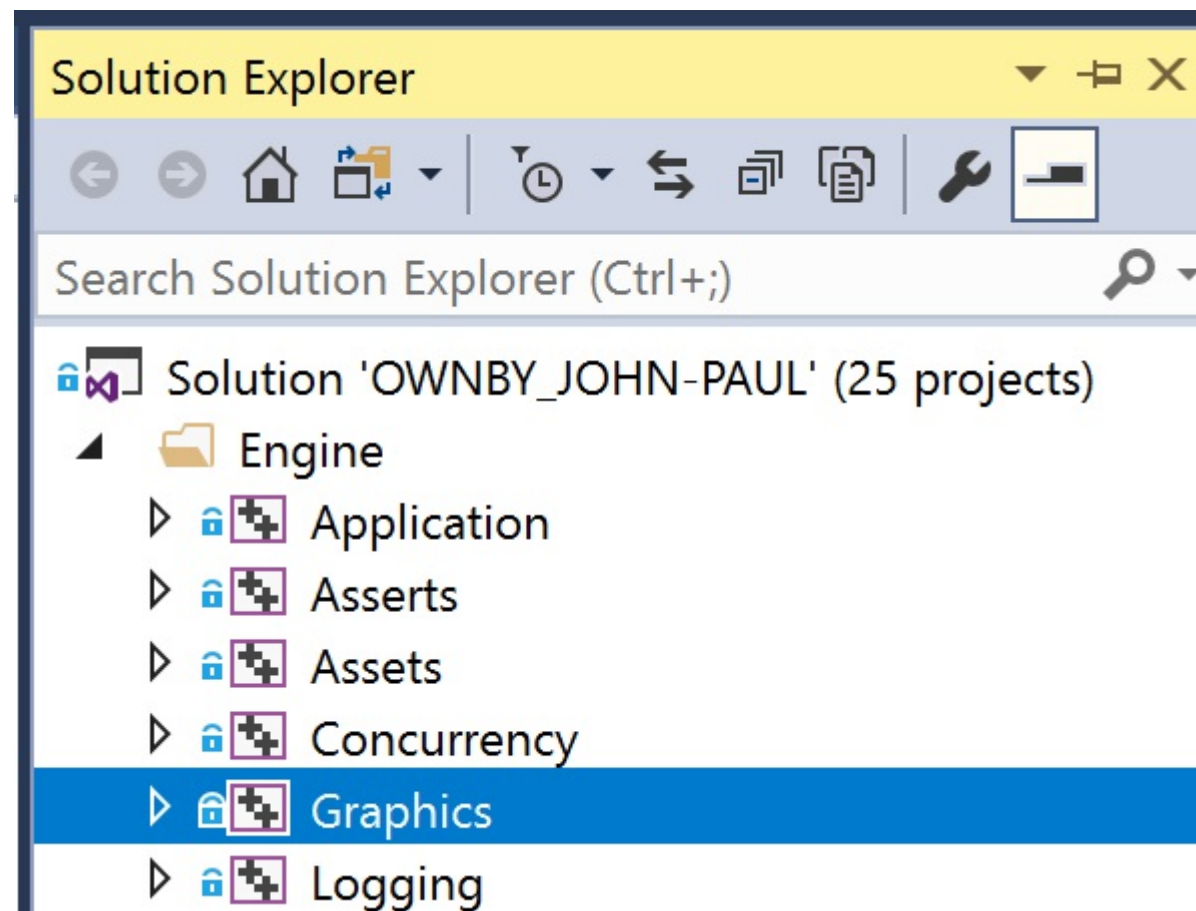
- The actual folder name (where the screenshot above shows "ReferencImplementation") can be anything that you want. Tony and the TA will never see it, because that is your git root directory. (When we download your solution from GitLab we will choose our own name on our own hard drive for it.)
- The actual solution file must match your GitLab project exactly, and so you won't be able to satisfy this requirement until your GitLab project is set up and you know what that name is
 - Once you know the name you can rename the SLN file using Windows Explorer


Graphics Library

- You must create a new Graphics project for your solution
 - It *must* be named "**Graphics**"
 - It *must* be located in "Engine" in your directory structure, like this:



- It *must* be within the "Engine" folder in Visual Studio's Solution Explorer, like this:



- Add it by doing the following:
 - Right-click the Engine folder in Solution Explorer and choose **Add->New Project...**
 - Select **Windows Desktop Wizard** and click Next
 - Enter "**Graphics**" for the Project name
 - Browse so that the **Location** is Engine/
 - Click **Create**, and a popup window will appear
 - Change **Application type** to **Static Library (.lib)**
 - Make sure that "**Precompiled header**" is not checked
 - Click **OK**
- The very first thing you should do with your new project is to add the property sheets
 - If you are not sure how to do this review the [Solution Setup Example \(https://utah.instructure.com/courses/886852/pages/solution-setup-example\)](https://utah.instructure.com/courses/886852/pages/solution-setup-example)
 - Remember that the order that property sheets are listed matters! Add EngineDefaults first to the entire project, and then add OpenGL to the Win32 configurations and Direct3D to the x64 configurations.
- Next, remove all of the files and folders that Visual Studio adds for you, and then delete them from disk
- Now, copy the [provided files \(https://utah.instructure.com/courses/886852/files/150334773/download?wrap=1\)](https://utah.instructure.com/courses/886852/files/150334773/download?wrap=1)  (https://utah.instructure.com/courses/886852/files/150334773/download?download_frd=1) into the Graphics library folder in Windows Explorer
- Then, add them to the project in Solution Explorer
 - Remember to set things up the way that the [Platform-Specific Code Example \(https://utah.instructure.com/courses/886852/pages/platform-specific-code-example\)](https://utah.instructure.com/courses/886852/pages/platform-specific-code-example) explains. If you are not sure if you have done it correctly open that example solution again and make sure that you have the platform-specific files organized correctly, and that they are set to only build with the appropriate configuration.
 - You will lose points if you don't have this correct!
- Now you need to set up the project dependencies. There are two parts to this:
 - Add the required references to your new Graphics project

- Usually you will be writing code and so you can add a reference when you need to use a new project. In this case, however, someone else has written the code and given it to you so you will have to do some detective work. Look through each source file in the Graphics project and see if you can figure out which other projects it calls functions from; these are the projects that you will need to add as references.
- Add your new Graphics project as a reference to any other projects that use it
 - There is a lot of code that you have been given, and so rather than looking through each file individually you can be more efficient
 - Search the entire solution (i.e. use Visual Studio's "Find in Files" functionality) for "**Graphics::**". That will give you a list of every line of code that mentions something from the Graphics namespace
 - Most of the results will be in the Graphics project itself, and you can ignore them
 - Some of the results, however, will be in other projects. Look for which projects call functions from the Graphics project. Those are the ones that will need a reference added.
 - Note that a reference (i.e. a dependency) is only required if a project calls a *function*, and even then only if that function is defined in a CPP file. Some of the results you will find when searching will be to things other than functions (e.g. enumerations). If a function call from a CPP isn't involved then you don't need to add a reference.
- If after completing this step you get LNK2019 "unresolved external symbol" errors then you probably have missed a reference somewhere. **DON'T PANIC!** Look at the error message, try to figure out what it's telling you (Which symbol is unresolved? What code is trying to use it?), and then see if you can fix it. If you are stuck review the [Static Library Example](https://utah.instructure.com/courses/886852/pages/static-library-example) (<https://utah.instructure.com/courses/886852/pages/static-library-example>) and start a discussion thread asking for help.
- If you have done everything correctly you should now be able to run the ExampleGame project (either from the debugger if the project's settings are set up as explained in the **Debugging** section in the [Solution Setup Example](https://utah.instructure.com/courses/886852/pages/solution-setup-example) (<https://utah.instructure.com/courses/886852/pages/solution-setup-example>)) or by double-clicking the executable in Windows Explorer in \$(GameInstallDir). You should be able to run both Direct3D and OpenGL in both Debug and Release configurations. Congratulations!
- Now that you have a working solution I **strongly** suggest making a git commit before going further in the assignment. That way if you mess anything up when manually editing Visual Studio files it will be easy to get back to a working state.

Personal Game Project

- You must use the example game to create your own game projects
 - Initially your personal game files will be almost exactly the same as the example game files, but you can't copy the files exactly because there are unique identifiers that Visual Studio uses for each project. You will need to create new projects with their own unique identifiers and then copy the example game files.
- Create a new Windows Explorer folder for your game
 - This must be named **MyGame_** (notice the underscore at the end!)
 - It should be a sibling folder to the ExampleGame_ folder
- Create a new "filter" in Visual Studio's Solution explorer for your game
 - This must be named **MyGame** (it doesn't need an underscore at the end)
 - It should be a sibling filter to the ExampleGame filter:
 - Right click the solution in Solution Explorer and choose **Add->New Solution Folder**
 - Name it
- Create a project for your game code:
 - Right-click the new filter you created in Solution Explorer and choose **Add->New Project...**
 - Select **Visual C++->Windows Desktop->Windows Desktop Wizard**
 - Click **Browse...** and change the **Location** to the new folder you created
 - It must be named **MyGame** (with no underscore)
 - Click **OK**, and a popup window will appear
 - Change **Application type** to **Windows Application (.exe)**
 - Uncheck **"Precompiled header"**
 - Click **OK**
- Close Visual Studio (if it asks if you want to save any files say yes!)
- Populate your new game project with starter files

- Find the new files that got generated in Windows Explorer
- Open the MyGame.vcxproj file in a text editor
 - If you don't already have a favorite text editor I would recommend **Notepad++** (<https://notepad-plus-plus.org/>) for this class. Any editor is fine, though (you may also just use notepad if you don't want to download or install anything).
 - You need to find the value of the following in the project file and save it somewhere:
 - **ProjectGuid**
- Make sure you have the ProjectGuid saved somewhere, and then close the project file and delete all of the generated files in that folder (so that the folder is empty)
- Copy all of the files in ExampleGame_/_ExampleGame/ to MyGame_/_MyGame/
- Rename the following files:
 - ExampleGame.vcxproj -> MyGame.vcxproj
 - ExampleGame.filters -> MyGame.filters
 - cExampleGame.h -> cMyGame.h
 - cExampleGame.cpp -> cMyGame.cpp
 - Resource Files/ExampleGame.rc -> Resource Files/MyGame.rc
- Copy and rename ExampleGame_/_ExampleGame.props to MyGame_/_MyGame.props
 - (Note that it is inside the MyGame_/_ folder with the underscore, but outside the MyGame_/_MyGame/ folder that contains the new project you have made)
- Open the new MyGame.vcxproj file (the one that you just copied and renamed) in a text editor
 - Change **ProjectGuid** to the new one that you saved
 - Search for all instances of "ExampleGame" and change them to "MyGame". This should be the "**RootNamespace**" and all of the files that you renamed.
- Open the new MyGame.filters in a text editor
 - Change all instances of "ExampleGame" to "MyGame". This should be the files that you renamed.
 - Find the "**Resource Files**" Filter
 - Since you copied from ExampleGame its current "UniquelIdentifier" isn't unique. You will need to make its UniquelIdentifier actually unique.
 - Generate a new GUID. I recommend using [this online tool \(https://www.guidgenerator.com/online-guid-generator.aspx\)](https://www.guidgenerator.com/online-guid-generator.aspx).
 - Copy and paste your new GUID to create a new UniquelIdentifier for the "Resource Files" Filter. (Make sure to keep the format of the GUID the same, {enclosed in curly braces}.)
- Open the new MyGame.props in a text editor and change the **GameName** from "ExampleGame_" to "**MyGame_**" (with an underscore at the end)
- If everything was correctly changed you should now be able to open your solution in Visual Studio with no errors and the MyGame project should look the same as the ExampleGame project. (Even though you should be able to open it without errors you won't be able to build it without errors yet!) If it doesn't work you can either look for errors that Visual Studio reports and fix them (this may or may not be possible depending on what the error is), or you could remove the project, delete the files, and start again. Don't be afraid to ask for help on the class discussion board if you run into problems!
- Once this part is working I would suggest staging your current changes in git. I personally wouldn't commit them yet (because your code won't build yet without errors), but staging changes is like making a quicksave in a game: If you make any mistakes on future steps you will be able to get back to this point.
- Update the file contents in the new MyGame project
 - There are three files that you will need to change: EntryPoint.cpp, cMyGame.h, and cMyGame.cpp (You shouldn't have to change any of the Resource Files)
 - You will have to change some #include directives
 - You should change the "include guards" in cMyGame.h
 - You should change the class name from cExampleGame to cMyGame
 - Once you have done this you should be able to build the solution without errors. You should then be able to find your new MyGame \$(GameInstallDir) folder in Windows Explorer, and it should contain MyGame.exe. You can even run it, but you will get errors because there are no shaders yet. Consider staging your new changes once this is working!
- Create an empty project to build assets
 - This will be very similar to how you created the MyGame project
 - Add a new project to the MyGame filter

- Select **Visual C++->Other->Empty Project**
- Name it **BuildMyGameAssets**
- Locate it inside the MyGame_ / folder (it will be in a sibling folder to the game project)
- Save the new ProjectGuid from BuildMyGameAssets.vcxproj, delete the generated files, and copy the files from ExampleGame_ /BuildExampleGameAssets/ to MyGame_ /BuildMyGameAssets/
- Rename the files that you just copied (you should be able to figure out what to rename them to)
- Open the new VCXPROJ file (that you just copied and renamed), and update it. You should be able to figure out what you need to change.
- (You shouldn't need to change anything in the FILTERS file)
- If you have done everything correctly you should now be able to open the solution in Visual Studio and the BuildExampleGameAssets project should look correct (you won't be able to build it without errors yet, though). Consider staging in git!
- Copy the ExampleGame_ /Content/ folder to MyGame_ /Content/

Set up the build dependencies for the BuildMyGameAssets project


- The BuildMyGameAssets project has some dependencies that Visual Studio can't figure out automatically
- You will need to look at which projects the BuildExampleGameAssets project depends on, and make the new BuildMyGameAssets project depend on those same projects
 - If you don't remember how to do this review the **[Solution Setup example \(https://utah.instructure.com/courses/886852/pages/solution-setup-example\)](https://utah.instructure.com/courses/886852/pages/solution-setup-example)**
- If you have done everything correctly you should now be able to build the solution without errors and then look at your MyGame \$(GameInstallDir) and see that it has data and Licenses. If you double-click the EXE it should now run without errors and show the white triangle, just like ExampleGame. Congratulations! You now have an engine that is used to build two different games! (Consider staging in git.)



John-Paul's EAE6320 Example Game -- Direct3D



o

- 
- Update the solution to make MyGame the default startup project
 - You can choose which project Visual Studio uses when you start the debugger. Sometimes you need to be able to switch between different projects to debug, but for our class it will be convenient to configure the solution so that the MyGame project is the default even if the user hasn't chosen any other project. Unfortunately Visual Studio doesn't provide any way to do this officially, but it is possible to trick it into doing what we want (sometimes the way to do so changes between different versions of Visual Studio, however).
 - (Make sure that any changes are staged or committed before doing the following steps! If you accidentally mess anything up it is important to be able to easily revert the bad changes.)
 - Make sure that your solution isn't open in Visual Studio
 - Open the solution file (e.g. OWNBY_JOHN-PAUL.sln) in a text editor
 - Notice that all of the projects are listed towards the top of the file. When you add new projects in the GUI Visual Studio will add them to this file in the same order, and so your new MyGame projects will be at the bottom of the list of projects.
 - Note that the filter you added is listed as a "project". You should see three MyGame projects listed (the MyGame filter, the MyGame project, and the BuildMyGameAssets project)
 - You need to move the MyGame filter and MyGame project to the top so that Visual Studio uses them as the default startup (you can leave the BuildMyGameAssets project where it is)
 - Cut and paste the two MyGame projects from the bottom to the top
 - Notice that the ExampleGame projects are currently at the top. You can use that as a guide to help you know how the file should look when you put the MyGame projects above them.
 - While Visual Studio is still closed delete the solution user settings
 - There will be a hidden folder in your solution folder called ".vs". If you don't see it you may not have Windows configured to [show hidden directories](https://support.microsoft.com/en-us/help/14201/windows-show-hidden-files) (<https://support.microsoft.com/en-us/help/14201/windows-show-hidden-files>).
 - Find the file .vs/[YOURSOLUTION]/v16/.suo and delete it
 - If you have done this correctly you can open the solution in Visual Studio, and the **MyGame** project's name will be **bold**. This means that it is the default startup project.
 - (Consider committing once this is all working before changing anything else!)

cMyGame.h Configuration

- You must change your game's window name
 - It can be anything you want, and doesn't have to include your name. (You should change it so that it doesn't have *my* name, though!)
 - Please the platform and debug configuration at the end like I have it so that it is easy to tell which version is running
- You should change your game's window class name, but this isn't necessary
- You should change the icon
 - My recommendation is to use one of the two EAE icons that I have provided (this is a trivial change that you should be able to figure out by reading the comments in the file). If you are interested, however, you can also try to create your own unique icon as described in the comments.
 - If you make your own icon make sure to tell us about it in your write-up!
 - Windows will sometimes cache icon files, and you may find that the icon shown in Windows Explorer doesn't change. If this happens, don't worry! As long as the correct icon shows up in the window when you run the game then you have done everything correctly.
 - If you create your own icon you will have to add it to ExampleGame/ResourceFiles/ and then update ResourceFiles/ExampleGame.rc and Resource.h in a text editor. It is easy to mess these files up accidentally, so make sure that you have any other changes staged in git before attempting this so that you can easily revert to a working version if something goes wrong!

Logging Messages

- It is a good habit to log messages about important things that happen during the game
 - This can help you figure out what went wrong after-the-fact by looking at the log file
- Any unusual errors should definitely be logged, but it is also useful to log some things that happen the way that they should so that you have a better idea when the errors happened

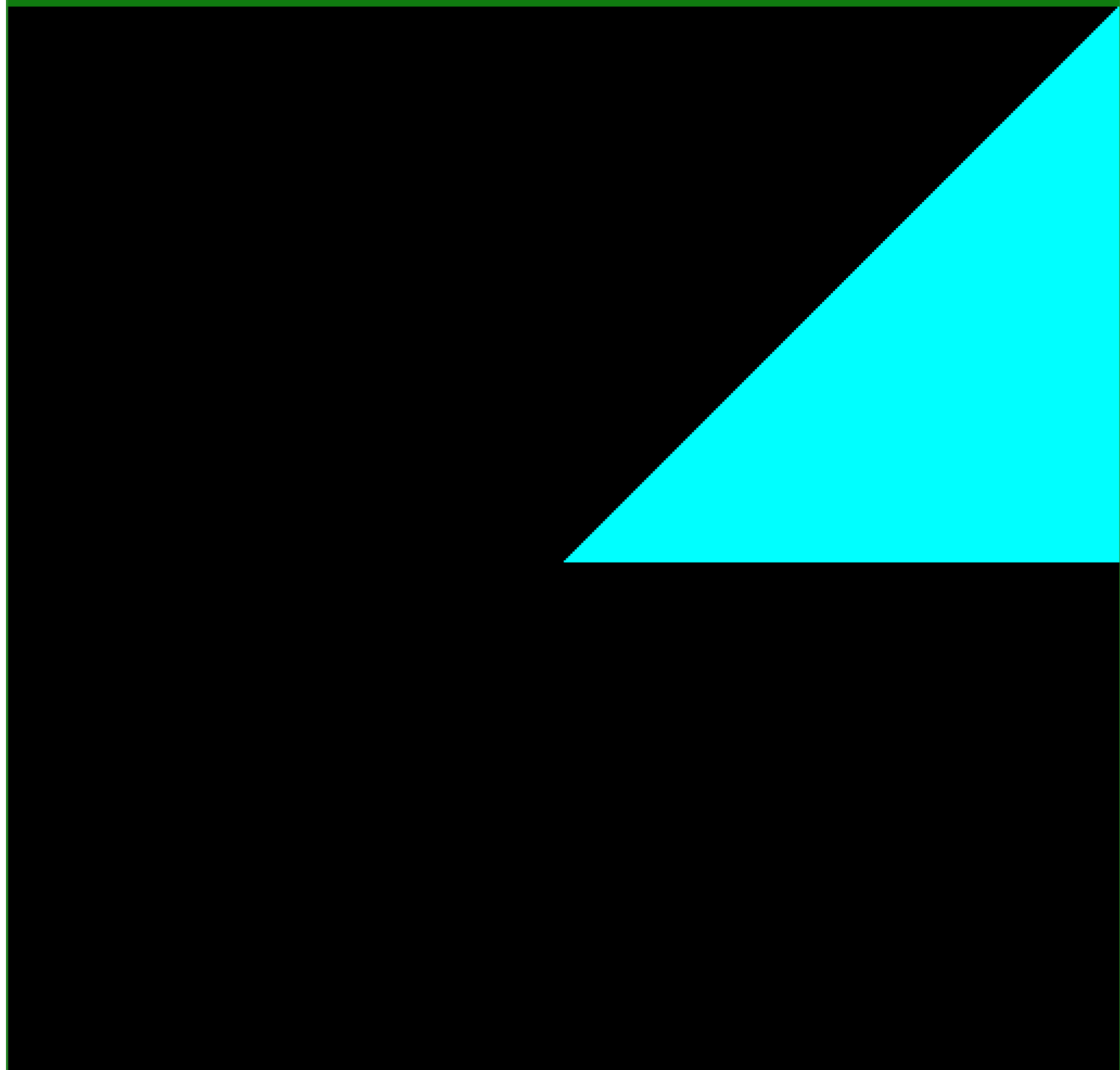
- I am not going to give explicit instructions on how to log a message
 - You should be able to look at other projects that I have given you for examples
 - If you get stuck you can ask for hints on the discussion board

Triangle Color Animation

- Create a new custom fragment shader for your game
 - Currently there are some "standard" shaders that are part of the engine because any game would need them. You are going to create a specialized shader just for MyGame.
 - Copy Engine/Content/Shaders/Fragment/standard.shader to MyGame_/Content/Shaders/Fragment/
 - You can choose whatever name you want (it shouldn't be "standard", though, since that doesn't make sense). For example, you could call it "animatedColor.shader", "special.shader", "myShader.shader", "test.shader", etc.
 - Open the AssetsToBuild.lua file
 - This should be listed in your BuildMyGameAssets project. You can also find it in your MyGame_/Content/ folder.
 - Add an entry for your new shader so that it will be built
 - Note that the existing entries are to the Engine shaders, but their paths don't indicate this. When given a path the asset build system will first look in the game's content directory, and if it doesn't find anything there will next look in the engine's content directory.
 - Copy the entry for the standard fragment shader, and change the filename to be whatever you chose
 - Build the solution and make sure that your new shader gets built without errors
 - Look in \$(GameInstallDir), and verify that your new shader is there
 - Change the graphics project to load your fragment shader instead of the standard one
 - You will have to do this in both Graphics.d3d.cpp and Graphics.gl.cpp
 - Find where the standard fragment shader is loaded and change the code to load your new shader
- Update your custom shader to output a color different from white
 - (Notice that there is separate code for Direct3D and OpenGL. Make sure that you are changing the same platform that you are testing!)
 - The output variable is called `o_color`. It is a float4 in Direct3D and a vec4 in OpenGL, which is a special shader type that is an array of four floats. When used as a color this is an "RGBA" value, which means that the first float is how much red ("R"), the second float is how much green ("G"), the third float is how much blue ("B"), and the fourth float is how much transparency ("A" for "alpha"). In our class we will only worry about the first three floats ("RGB").
 - Each of the floats should be a value between 0 and 1, where 0 means none and 1 means full. So, for example, red would be represented as (1,0,0), green as (0,1,0), and blue as (0,0,1). White is a combination of all colors (1,1,1), and black is the absence of all colors (0,0,0).
 - There is special syntax to change individual floats. You can use "r", "g", and "b" to index each color "channel". So, for example, if you add the following line to the shader you will change o_color from (1,1,1) to (0,1,1):
 - ```
o_color.r = 0.0;
```
  - If you try that your triangle should look like this:



John-Paul's EAE6320 Example Game -- Direct3D

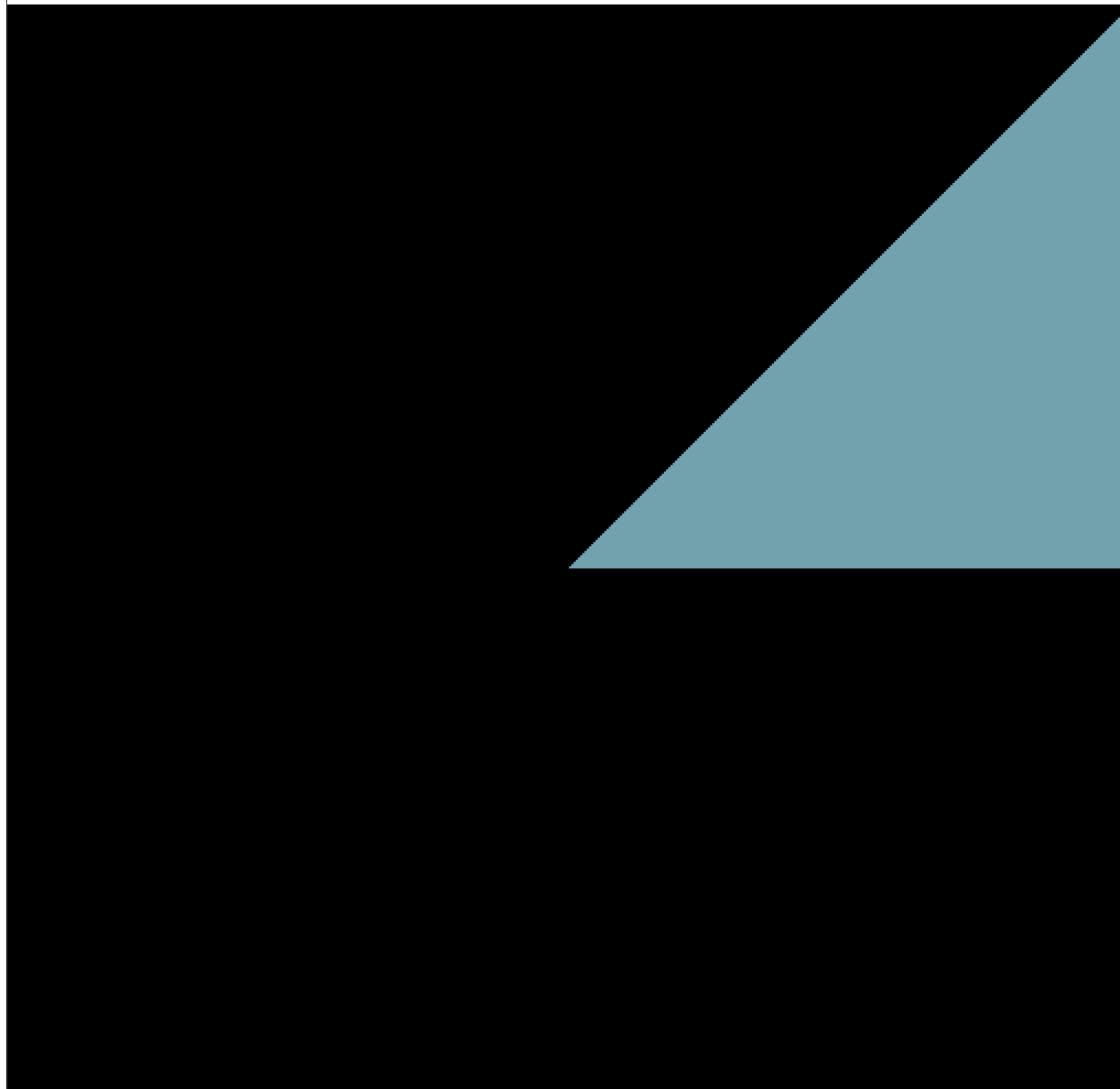





- Similarly, you can change green with `o_color.g` and blue with `o_color.b`. You should try experimenting with different value for the three color channels (remember that each float can be anything between 0 and 1).
- Add animation so that the color of the triangle changes as time passes
  - This is your chance to be creative! As an example here is what I have in my reference implementation (you don't need to try and match this, however):



John-Paul's EAE6320 Example Game -- Direct3D



- 
- The value `g_elapsedSecondCount_simulationTime` constantly changes as the simulation progresses, and so you can get a value that oscillates between [-1,1] by doing something like:
    - `sin( g_elapsedSecondCount_simulationTime )` or `cos( g_elapsedSecondCount_simulationTime )`
  - With that in mind figure out a way to animate the color of the triangle as time passes. Remember that the final value of each float in `o_color` should be between 0 and 1, and so you may need to do some math.
  - If there is an error in a shader you should see a Visual Studio error message telling you about it. In Direct3D double-clicking the error message should take you to the source shader. In OpenGL, unfortunately, double-clicking the error message will take you to the *built* shader. You will have to look at the error, and then remember to change the source shader in `$(GameSourceContentDir)`. If instead you accidentally change the version in `$(GameInstallDir)` then that change will be overwritten as soon as you build the BuildMyGameAssets project again.
  - Remember that both platforms must work identically! Once you are happy with the color animation on one platform make sure to add it to the other platform and test it.
  - If you get confused or have trouble with this requirement then don't hesitate to ask questions on the discussion board!

## Optional Challenges

- Each of the four example shaders re-declares the constant buffers. Can you figure out a way to only declare them in one place so that you don't have to duplicate code?
- Can you figure out a way to allow the user to press a key and either pause the game or make it enter slow motion while the key is held down (and resume normal operation when the key is not held down)? There are a few things you will need to figure out to get this to work:
  - How does keyboard input work in the engine I've given you?
    - The code I've given exits the application when the ESCAPE key is pressed; if you can figure out how this works it will be a hint. (One difference is that pressing the escape key updates the entire application; your challenge is to just update the simulation.)
  - How does time work in the engine I've given you?
    - The time that you want to stop or slow down is "simulation time" (not "system time"). Can you figure out how to manipulate the rate that the simulation is run at? (There is already a way to do this; you just need to find it and then set the simulation rate based on key presses.)