Assignment 02

The Canvas assignment is **here (https://utah.instructure.com/courses/886852/assignments/12883236)** .

Requirements

- Create a representation of a mesh
  - There must be a platform-independent interface for:
    - Initializing and cleaning up the mesh
    - Drawing the mesh
- Replace the geometry data in Graphics.d3d.cpp and Graphics.gl.cpp with an instance of your mesh representation
- Create a representation of an effect
  - There must be a platform-independent interface for:
    - Initializing and cleaning up the effect
    - Binding the effect
- Replace the shading data in Graphics.d3d.cpp and Graphics.gl.cpp with an instance of your effect representation
- Add a second triangle to the mesh so that a rectangle is rendered
- Your write-up should:
  - Show a screenshot of your game running
    - It should show a rectangle in the upper-right corner (the color will depend on your color animation)
  - Show screenshots from a Direct3D GPU capture in the Visual Studio Graphics Analyzer:
    - The first screenshot should show the render target when it is all black (the ClearRenderTargetView() function should be highlighted)
    - The next screenshot should show the render target with the mesh (the Draw() function should be highlighted)
      - Also, the Pipeline Stages tab should be selected, and the mesh should be visible as "wireframe", where the two triangles are visible
  - Show screenshots from an OpenGL GPU capture in RenderDoc:
    - The first screenshot should show the render target when it is all black (the glClear() function should be highlighted, and the Texture View tab should be selected)
    - The next screenshot should show the render target with the mesh (the glDrawArrays() function should be highlighted and the Texture View tab should be selected)
    - The final screenshot should show the mesh's two triangles (the glDrawArrays() function should be highlighted but the Mesh Output tab should be selected, and either the VS Input or VS Output sub-tab should also be selected)
  - Show your code that binds the effect and draws the mesh in either Graphics.d3d.cpp or Graphics.gl.cpp (the code should be identical in both files, and so it shouldn't matter which file you show; if the code is different in the two files it means that you haven't completed the assignment correctly)
    - Specifically, this should show the platform-independent interfaces being used. **Do *not* show your implementation code**; show us how the interface that you have designed is easy to use and platform-independent.
      - (Do *not* show code from your mesh or effect CPP files! Show code from Graphics.d3d.cpp or Graphics.gl.cpp that *uses* your interfaces.)
    - You can "show" code by either taking a screenshot and cropping it or copying/pasting the source code as text. This particular requirement should only be a few lines! If you have more than a few lines you are either not showing what this requirement is asking for or there may be improvements that you should make in your interface.
  - Tell us (in general terms) what the remaining differences are between Graphics.d3d.cpp and Graphics.gl.cpp. Discuss any ways that you can think of to use platform-independent code to eliminate those differences so that only a single platform-independent Graphics.cpp file would be necessary.

Submission Checklist

- Your write-up should follow the **standard guidelines for 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)**

- Are your new platform-specific effect and mesh files named correctly? Are they in the correct location on disk? Are they in the correct location in Solution Explorer?

Finished Assignments

Details

Mesh

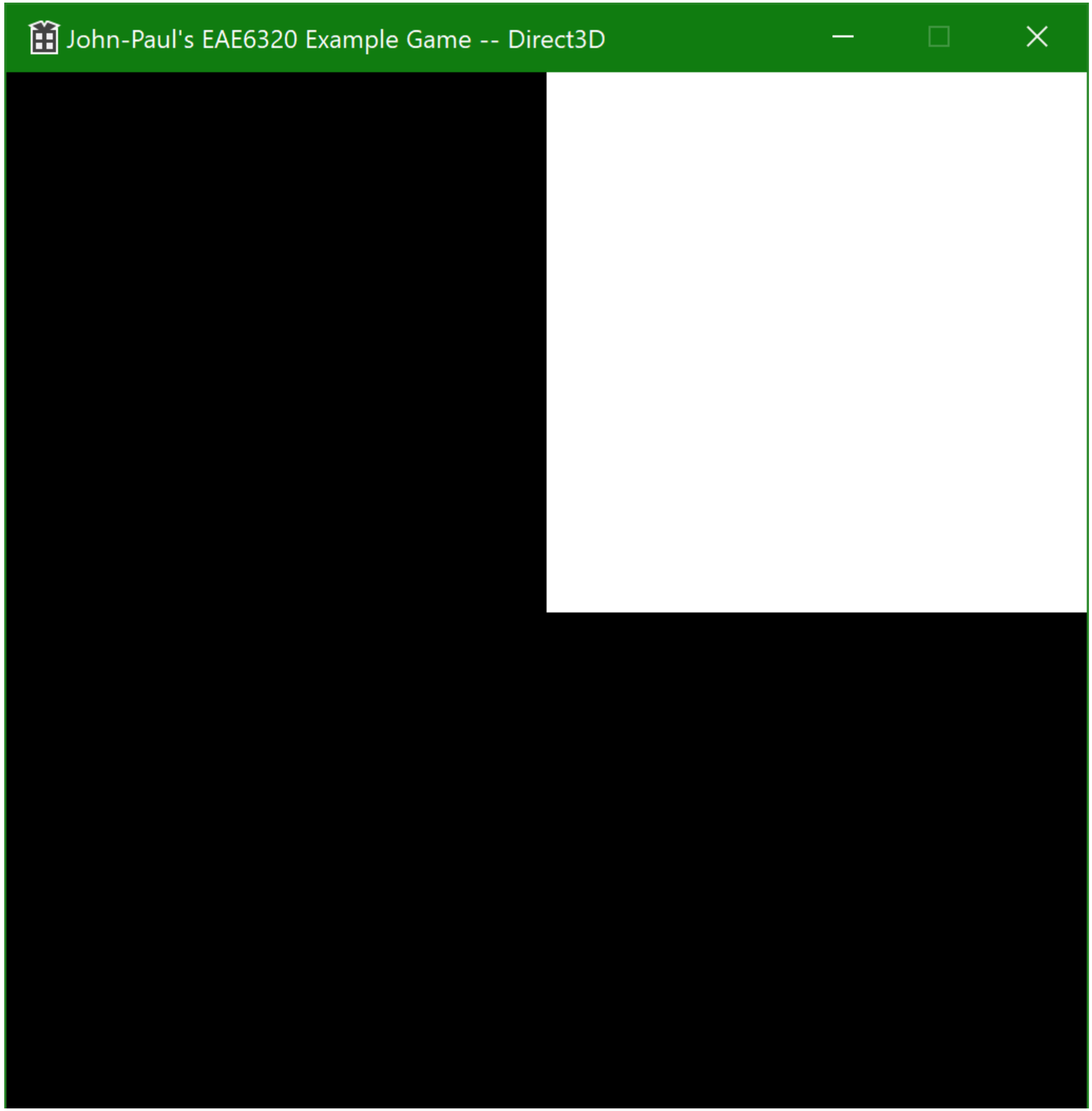
- You should make either a struct or a class, and you must follow the convention of where to place platform-specific implementation files and what to name them (refer to the **Platform-Specific Code Example (https://utah.instructure.com/courses/886852/pages/platform-specific-code-example)** for a reminder)
  - Other than those two rules, however, you are free to use your own coding style and conventions
  - (If you wish you may choose to follow the example classes of mine or try my coding style. If you do then when you make member variables please use the "m_" prefix to denote "member variables" rather than keeping the "s_" prefix which denotes "static variables" or you will drive me crazy :)
  - When deciding what to name your representation keep in mind that it only represents a 3D object's *geometry*, and is internal to your Graphics system. Later in the semester you may want to have a different representation of an actual 3D entity that your game uses (that includes this mesh in addition to other things).
- The main thing you should be doing for this requirement is moving code from Graphics.[platform].cpp into your new mesh CPP files (either the platform-independent one or the platform-specific one). You may choose to rename things or change formatting to fit your style, but my recommendation would be to first just move code and get it working. Once you do that, make a commit, and *then* refactor.
- The data that belongs in a mesh can be found in the Graphics.[platform].cpp files under the "Geometry Data" comment. You will have to look at that data (for both platforms), and then look at how it is drawn during rendering, and how it is initialized and cleaned up. Your task is to create a struct or class to represent the concept of a "mesh" and then move the platform-specific data and functions into this struct/class so that they can be used with a platform-independent interface.
- Keep in mind that your game will eventually have more than one mesh. When designing your representation think about what would happen if you tried to create a second mesh.

Effect

- You should make either a struct or a class, and you must follow the convention of where to place platform-specific implementation files and what to name them (refer to the **Platform-Specific Code Example (https://utah.instructure.com/courses/886852/pages/platform-specific-code-example)** for a reminder)
  - Other than those two rules, however, you are free to use your own coding style and conventions
- The main thing you should be doing for this requirement is moving code from Graphics.[platform].cpp into your new effect CPP files (either the platform-independent one or the platform-specific one). You may choose to rename things or change formatting to fit your style, but my recommendation would be to first just move code and get it working. Once you do that, make a commit, and *then* refactor.
- The data that belongs in an effect can be found in the Graphics.[platform].cpp files under the "Shading Data" comment. You will have to look at that data (for both platforms), and then look at how it is bound when being rendered, and how it is initialized and cleaned up. Your task is to create a struct or class to represent the concept of an "effect", and then move the platform-specific data and functions into this struct/class so that they can be used with a platform-independent interface.
  - Some of the data and functions are platform-specific, but some of them are platform-independent. Whenever it is possible to make something platform-independent you should! This makes it easier to understand and maintain code.
  - To be absolutely clear, you should have three CPP files associated with your effect representation (for example, if you make a class and follow my naming convention you would have a cEffect.cpp, cEffect.d3d.cpp, and cEffect.gl.cpp files), and you should have some platform-independent implementation code in your effect struct/class platform-independent CPP file.
    - (Your mesh implementation will likely be almost entirely platform-specific, but only *some* of your effect implementation has to be platform-specific. Try to find anything that can be shared make it platform-independent! This will make your code more understandable and maintainable.)
  - There are three pieces of functionality (draw, initialize, and clean up) that you need to implement, and you may find that some parts may need to be platform-specific but other parts can be platform-independent. Can you figure out how to do this?
- Keep in mind that your game will eventually have more than one effect. When designing your representation think about what would happen if you tried to create a second effect.

Second Triangle

- There are two things that you need to do to get this to work:
  - Update any code that says there is 1 triangle to now say that there are 2
    - If you have done the assignment correctly this code should all be encapsulated in your mesh representation
      - (If it isn't fix your representation so that it is!)
    - Once you have made this change your code will be incorrect until you complete the next step
  - Add more data to the vertex array used to create the vertex buffer
    - In the code that I gave you that you used for Assignment 01 there is an array of 3 vertices used to create the single triangle. You will need to add 3 more vertices to this for the second triangle.
    - You should have an understanding of how to do this based on the in-class lecture. The most important thing to remember is that Direct3D and OpenGL have different "winding orders", which means that the actual vertex data that you add to each platform should be the same, but the order that you add that data for the second triangle will be different!
      - Remember that Direct3D uses a left handed winding order and OpenGL uses a right handed winding order
      - Don't be afraid to draw out the vertices and triangles on a piece of paper! If you can figure it out in your head then that's fine, but it will be *much* easier to draw it out and see it visually, especially if you don't have previous graphics experience.
- (Note that you don't have to make any shader changes)
- If you successfully add a second triangle your game should look like this (although the color will probably be different because it should be animating):


John-Paul's EAE6320 Example Game -- Direct3D

GPU Captures

- The **GPU Captures (https://utah.instructure.com/courses/886852/pages/gpu-captures)** page should have the information you need to figure out how to do this
- This requirement is to help you (so that you will start learning how to debug graphics problems in future assignments), and is not intended to make you confused or spend a lot of time. If you run into any issues with GPU captures don't waste time being stuck! Post to the discussion board and we will try to help you figure out what to do.
- In previous years some students have never been able to get captures to work (usually because the capture program crashes). Please try to get things working (since knowing how to debug graphics problems can help you in future assignments), but if you have posted to the discussion board and still can't figure it out send me an email and we can discuss what to do.

Optional Challenges

- Can you figure out how to make a house shape with triangles? You will probably have to use more than two. If you make a house or anything else cool besides a rectangle that uses more than one triangle include a screenshot in your write-up.