Grading Check List for GAM 398/598

This document lists some of the main features and tasks that you are expected to have completed for your final submission in GAM 398/598. Please consider using this document both as a guide for your final to-dos ***and as a check-list to use while recording your final video report***.

General Notes:

- Your projects are expected to compile without errors. A non-compiling project is worth zero and will not be inspected further.
- Your projects are expected to compile without warnings. All warnings must be fix/addressed, whether C++ warnings or DX warnings. Setting the compiler to ignore warnings is not acceptable. (Missing .pdb warnings can be ignored)
- All member variables should be accessed through setter and getters.
- You should use basic asserts in your code to detect invalid situation (invalid mesh number, for example)
- Many classes have changed/evolved over the course of the weeks (Ex: StandardVertex, etc.). You are expected to use and/or have implemented the latest version with the latest requirements

Final Scene

A demo scene where you will showcase all/most of the elements we covered this quarter. This scene must include:

- Skybox, Terrain, various premade models, some 'file loaded' models and light sources.
  - It's fine to use some of our pre-made models, but you are expected to make use to nicer looking models from file, either made yourself or found on the internet.
  - Note the models place this way are expected to be to scale and properly colored and/or textured. *This may require you to edit models for scale and/or alignment before converting them to Azul format*.
- There must be some attempt at making the scene look aesthetically interesting. By definition, this is a subjective criterion. I'm not asking for high-art, but if it looks like you slapped it together at the last minute, you will lose points. A Lot.
  - Issues such as: misaligned textures, scaling issues, 'random' objects without reason, etc. are all potential deductions. If in doubt, show me your scene (or send a snapshot) and I'll review it.
  - Your scene worthy to be placed in your portfolio to prove your knowledge of graphic environments.
- The scene should have multiple light sources:
  - This means all (most?) objects in the scene *must* respond to lighting effects.
  - Light effects should make sense: For example: a skybox should not in general interact with light sources. It may however interact with the fog. Obviously, if the fog completely obscures the skybox, that's not good.

        - At least one directional light (there is rarely a need for more than one anyway)
        - Multiple (3+) point sources OR multiple (3+) spotlight sources. Combinations of more light sources are fine too as long as one type has 3+ sources.
        - At least one light source should be moving in some way
    - Some elements should be moving to better showcase lighting effects.
        - The movement should not be gratuitous but actually justified within the scene's theme/context
    - Make judicious use of the fog effect and (optionally) mirror effect. See previous section regarding requirements for using mirror.

Expected Code Elements/Features

- Premade Models:
    - Should include at minimum unit box, pyramid and sphere
        - Should exist in multiple version of face texturing options
    - May also include other models (such as those used by FlatPlane, Skybox, etc.)
    - All these models should make use of the latest version of StandardVertex and set all the relevant values (uv, color, normal, etc.) when appropriate.

- Model Class:
    - Should have constructors for: Premade models, models from file and using arrays of StandardVertex and indices.
    - Should have the necessary mechanisms to render the model either as a whole or only a specified mesh.

- SkyBox Class:
    - The initial requirements for the skybox called to only have the proper model build to scale. However it would make more sense if it operated like its own specialized graphic object. Specifically: On construction, it should receive:
        - A pointer to the texture shader
        - A pointer to the texture to use
        - The size of the skybox
    - Calls to rendering should deal with the shader data as well as render the model correctly

- TerrainModel Class:
    - Should build a terrain model to scale according to a heightmap texture.
    - Should assign uv-mapping with specified repeats
    - Should also assign vertex normal *as the average of the incident face normals*.

- Terrain Class
    - As with the Skybox above: it would make more sense to have Terrain class acting somewhat like a graphics objects: managing pointers to shaders and textures and dealing with rendering in a more streamlined way.

- Shader Classes and associated HLSL shader files
    - Two broad types: color and texture based
    - All shaders are expected to
        - correctly work with a perspective camera
        - Implement the Phong Illumination model in the pixel shader with multiple light sources
            - Note that for the final scene, you are expected to have multiple lights of the same type (point or spot lights)
        - Implement the simple spherical fog with proper method for user-given parameters
    - **IMPORTANT:** Since shaders will control many settings, the user may not need or want to set every one of them. It is critical that your shader have reasonable default values for all settings.
        - **Ex:** If no fog parameters are provided, there should be no visible fog effect on the scene.

- Graphic Object classes corresponding to shaders for color, texture and their light version
    - Constructor receiving pointers for shader and model
    - Various setters for the many properties relevant to the associated shader
        - Most of these properties should be set on a per-mesh basis
    - Rendering method automating the required shader and model operations

- Mirror
    - (Required) As part of assignment 8: a working mirror with texture and multiple lights
    - (Optional) If incorporating a reflecting surface in your final scene (calm water, an actual mirror, a metallic object reflecting the scenery, etc.) the you need to create a mirror class that automates some of the steps. Specifically, this class should
        - Hold and manage the multiple pipeline state necessary for the mirror illusions
        - Compute and give access to the correct reflection matrix
        - A single method call (at the appropriate point in DrawScene) should
            - Deal with marking the stencil
            - Prepare the pipeline for drawing the reflected objects
        - The use is responsible to follow-up that call with drawing all the reflected objects
        - Another mirror method should cancel the reflected mode and render the mirror itself (and return the pipeline to its normal state)

- DXApp Changes
    - Your final projects should have the code in DXApp significantly streamlined. In particular:
        - DXApp::InitDemo should be essentially composed of asset loading (models, shaders and textures) and GameObject initialization.
        - DXApp::DrawScene should be mostly shaders being set to context, common shader data settings and calls to graphic objects' render method.