**Advanced Games Programming – Assignment 1**

# Abstract:

This document represents the group and individual work of Nedelin Gochev and George Alexandru Ciobanita for the first assignment in Advanced Games Programming. The first parts in this document represent the group work of the students while the following will represent each student’s individual work and conclusions according to it. The team will also offer its conclusion at the end of the document.

# Game Overview:

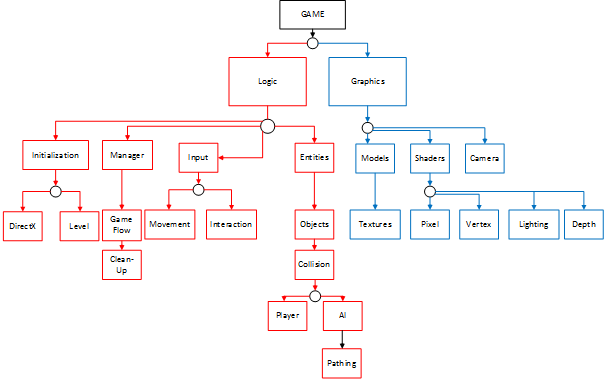
The team’s proposed game is the creation of a “museum” like environment where they can showcase different elements/requirements of the assessment in the environment. This can be done in either separate rooms or a single larger room. The environment will be used to demonstrate the following:

* Camera:
  + The game will be played from a first person perspective, similar to a First Person Shooter.
  + The camera is attached to the player head(or where the head would be)
* AI:
  + “Visitors”, who are wandering characters with random/specific paths and will try to avoid blockers on their path
  + A following entity, possibly for the resemblance of “a dog” that follows the player
  + An entity that on player collision will promptly apologize and go on a new walking path
* Exhibits:
  + Will be used to demonstrate different areas of study: lighting, shadows, collisions(also available in other parts of the game) and/or different kinds of shaders
* Collisions:
  + Collisions will be available through the environment
  + These will be used sparingly, avoiding their use in areas where players shouldn’t be able to reach
* Interaction:
  + The player will be able to interact with various exhibits in the museum
* Graphics:
  + There will be entities with different models in the environment (exhibits, objects, entities, etc.)
  + Different types of shaders will be used, from more simpler ones such as Pixel/Vertex shaders to possibly more advanced ones made by the team
  + The environment will feature lighting and shadows

This leaves the project open to possible additions if the time allows it. These can be:

* Sound, both for the environment and its entities
* Procedural generation of rooms
* Animations

# Game Breakdown:



**Figure 1:** *Breakdown of the team and its features in a format similar to classes with their different necessities, done as a group.*

The game was split into reasonable sized chunks and elements deriving from each other. Considering previous experience (Computer Games Assignment – Develop a 2D Game) the team has decided that the graphical side (which is also a more DirectX intensive side) will require more work and time to proper develop.

Considering the above, the “Logic” side of the game, while covering more aspects of a functional game, may require less or equal amounts of time and effort comparatively to the “Graphics” side.

A single programmer cannot cover multiple areas of study, especially on big projects. We already see in the industry that there are different areas of specialization: gameplay programmers, AI programmers, graphics engine programmer, sound programmer, etc.

Using the above information as a basis the team has agreed to split the work as follows:

* George Alexandru Ciobanita will take the “Logic” part described above
* Nedelin Gochev will take the “Graphics” part described above

This way, members can focus on specific areas of study and research, without fear of confusion from covering too many areas. Thus members can provide proper information to each other and communicate new findings in a better manner.

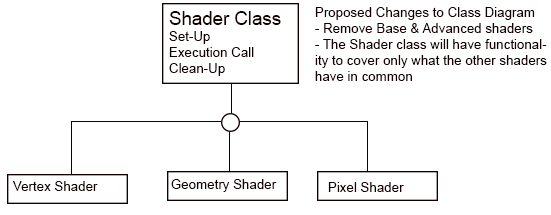
For a basic view of the team on class diagrams and game flow please refer to **Figure 2** and **Figure 3** in the Appendix.

# Version Control & Log Keeping/Testing:

The following are proposed for better and safer work conduct:

1. Github, in order to keep files, versions and backup available online.
   1. Team members will upload files and information, with appropriate descriptions and titles, as they acquire it and keep it available in the “AGP-Assignment” repository.
   2. Members can create/update files as they see fit (there is a document available in the repository that each member has to update).
   3. Members will communicate with each other in order to clearly update any current piece of information. There is the possibility that the findings of one member can help the other realize mistakes or improve.
2. Visual Studio can be setup to create logs after the codebase has been compiled and tested.
3. For future use the team can employ the use of “assert” a function mainly used for testing. This can work well with the Visual Studio log implementation as the following are logged: the function expression, name of source file, and the line number where it happened (e.g. Assertion failed: *expression*, file *filename*, *line number*).
4. Grey-box testing, a combination between White & Black box testing, can be used through implementation.
5. The tutorials available from Advanced Games Programming give a good view in how to setup a project in order to properly get error/warning messages when certain systems do not function. This can be specifically seen in Tutorial 01 and 02.

# Individual Design



When the team started designing this part of the software, we were working on only a theory on how rendering and shaders work backed-up by no research and assuming that it works the same way as the rest of the software we have developed so far. After researching the subject, the previous lack of knowledge became obvious. First of all a change to the class diagram of the shaders is proposed. The Base & Advanced shaders are not needed at all. The shader class will contain some basic functionality such as set-up & clean-up that are common for shaders in DirectX.

## What is a Pixel Shader(PS)?

PS is a program that runs on the GPU. The PS computes data for a single pixel but to do that it can take and use data from and about the neighbouring pixels. Effects such as blur, anti-aliasing, gradients, lighting and more are achieved with this shader.

## What is a Vertex Shader(VS)?

VS is a program that runs on the GPU. The VS takes data about vertices and manipulates them then outputs the results. . A VS is ran once for each vertex fed to the GPU. There can be various effects that can be created by the VS such as colour value changes, modifying texture coordinates, take 3D coordinates and map them onto a 2D projection plane. Note: This shader can only manipulate vertices, it cannot create new ones.

## What is a Geometry Shader(GS)?

This shader is executed after the VS. The GC, in contrast to the VS, can create new vertices and primitives. This shader allows meshes to be smoothed by adding extra geometry to them. For example, it can take a data of a cube and add vertices until it becomes a sphere. Another example would be generating extra lines for a series of lines that are forming a very crude arc to smooth it.

## Rendering Pipeline

DirectX processes and executes data and Shaders in a fixed order. It splits the work into stages. Counting only the stages that are used in the designed application so far, the order goes as:

1. Input-Assembler – this takes data such as points, lines & triangles and assemble them into primitives that would be used by the following stages.
2. Vertex Shader – Performs “Per vertex” operations (see above). This stage must ALWAYS be active for the pipeline to execute.
3. Tesselation – Converts low-detail primitives into high-detail and vice versa for GPU computation purposes
4. Geometry Shader – Manipulates Vertices(see above)
5. Stream Output – Streams data into the GPU memory for use in later stages
6. Rasterizer – Converts the 3D data from the above stages into a 2D data of pixels
7. Pixel Shader – Per-pixel data manipulation. (see above)
8. Output-Merger – Merges the data on the memory from the previous stages and generates the final data that would be outputted on the screen

## Software Design

Models Class

This class will hold data for 3D models used in the game.

Texture Class

This will hold data for 2D textures used in the game.

Note: The above classes may be totally unneeded as the DirectX engine may work with its own classes.

Shader Class

Will be used as a parent for the creation of all other shader classes.

Will have the basic functionality for shaders such as set-up and clean-up.

Will work as a manager for the shaders.

Camera

This will work as a manager for the game camera that will be attached to the player character.

It will hold the data needed for the camera to work properly such as:

* Position
* Facing
* Rotation
* Field of View
* Draw distances
* Clipping

Also, it will hold methods to manipulate the above data.

Additional Shaders

Addition of all other techniques that we are going to learn during this course is considered with a very high chance. Additional shaders may be implemented such as lighting shaders or tessellation shaders. As research has shown, Lighting shaders are mainly more advanced pixel shaders and Tesselation shaders are closely connected to Geometry shaders.

# 5. Conclussion

5.1 Group Conclusion

After sitting down and discussing the different information what was gathered and left available on GitHub, the team realised that a substantial amount of time will be dedicated to setting up DirectX and getting the different parts of the engine working. This includes using different kinds of shaders, which in turn need to be created by the student, setting up lighting, a proper camera and everything else that is more graphically intensive.

Of course the line is not drawn there as the team wants to show signs of improvement from last year’s assignment, Computer Games Programming Assignment 2, and show improvement code-wise when it comes to making a more object oriented approach for the classes and how the game functions, in order to attempt poor and slow implementations of code.

It is agreed upon that different means of future-proofing the project need to be set in stone in order to more easily test and debug a project of this size and scope, as the time available for work is limited.

5.2 Individual Conclusion

The first design is based entirely on theory with a wrong idea of how the DirectX pipeline & rendering works. After a research on the subject the flaws in the design have become obvious. Still, the re-design is entirely theoretical, but this time it is based on a research. Proposed changes would be to:

* Create a manager class for the shaders that works as a plug-in and set-up for the shaders
* The “Base Shader” & “Advanced Shader” classes are not needed at all.
* Some of the shader classes in the class diagram are actually deriving from each other, e.g. Pixel & Lighting shader
* The amount of shaders that can be put into the game & their complexity depends on the extra time that is left for implementation and may vary.