# Intro

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.

# Pixel Shader (PS)

## 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.

# Vertex Shader (VS)

## What is a Vertex Shader()?

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.

# Geometry Shader (GS)

## What is a 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

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

## Textures

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

## Shader Class

Will be used as a parrent for the creation of all other shaders classes.

Will have the general functionality of the shaders such as initialization and claen-up.

Will work as a manager of the shaders and effects.

Will have two child classes – Base Shaders & Advanced 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.

# 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.