Real-Time Graphics / C++ Programming and Design

RT_GFX Report

Submitted for the or MEng in
Computer Science

January 2017

by
Nick Ignat Smirnoff

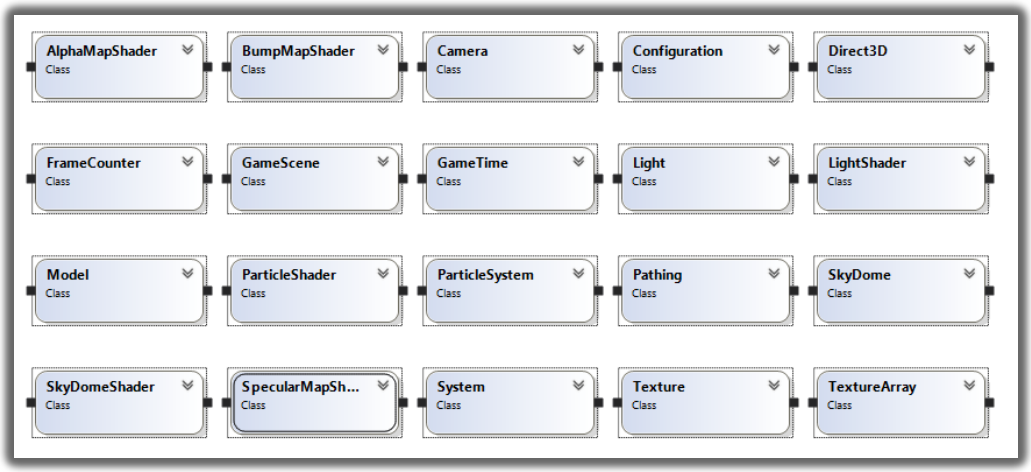
Word Count: 2369

Contents

1 Introduction

This document will outline the design and graphics, with project management of the single portfolio work “RT_GFX”. This project is a simulation of an under-water environment captured within a sphere, submarines and sea life move around the water world; this substantial piece of software demonstrates my ability and knowledge of 3D computer graphics that I have acquired over the time of the project.

2 Design



Screenshot of the project class diagram within Visual Studio.
View Appendix A for expanded view.

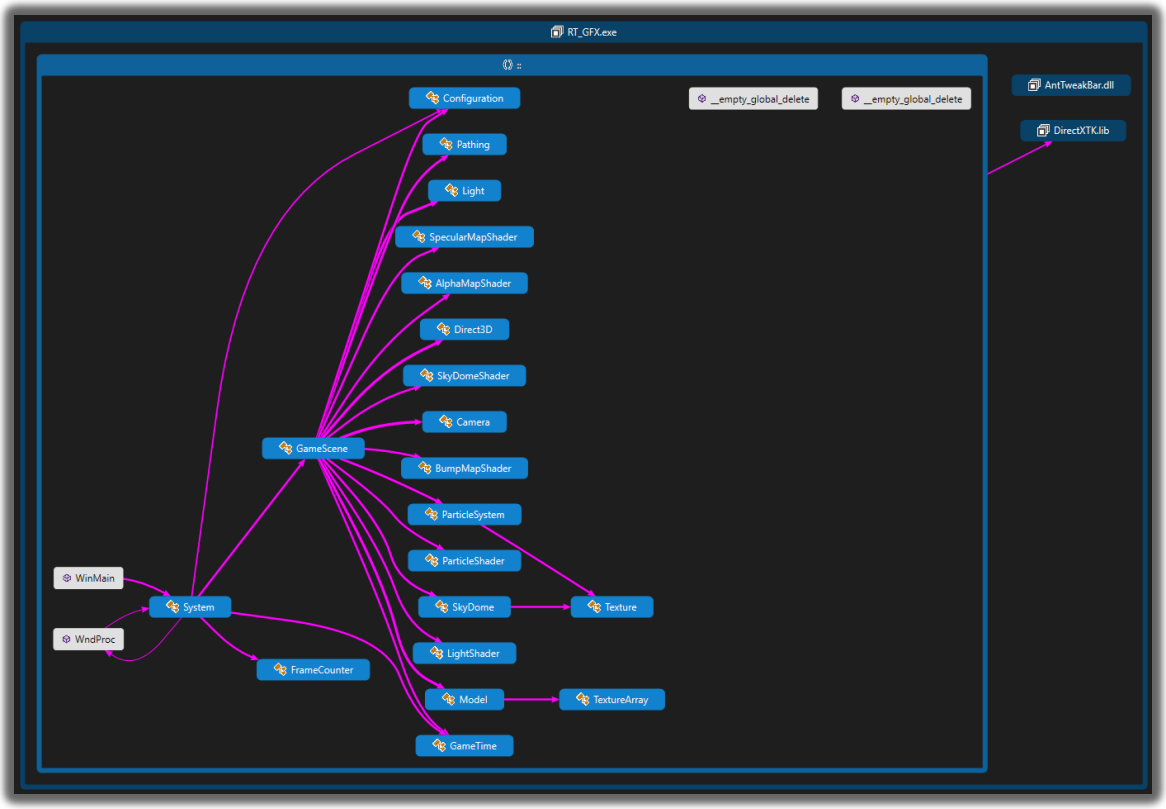


Image of the project class diagram.

Class Description

"WinMain" initiates the "System" class in which the basic settings from the "config.cfg" file are read in and contribute in making the basic window, upon window initialisation the main system loop is initiated in which "GameScene" is initialised and updated.

The system loop will carry on updating until "done" is given to be true, this process happens through the intention process of the user hitting the escape key. With each loop the "FrameCounter" is updated detailing frame values.

"GameScene" is the main graphics class and carried out the initialisation of the objects within the program, upon initialisation "Direct3D" is initialised, followed by the "GameTime", "Camera", "Model", "Pathing" and the remaining lighting and shaders classes.

The "Model" class initialises models through the input of parameters linking to the models structural data values (text file), and the texture files (image file) loaded through "TextureArray" as the techniques used require multiple texture files. Textures are loaded through a third-party "DirectXTK". Each model when successfully initialised will be given a position and if necessary a path, both acquired through the configuration file.

The "Camera" is updated through an input method which is called every frame, this input method handles the controls associated with the program and is mainly used to interact with the camera, with an exception to resetting the application to its initial state and changing render states.

Rendering will use the desired shader methods depending on the object or model, these techniques will use values from the "Light" class to function appropriately. Upon completion of the system loop all shutdown method will be carried out in which all the object pointers used will be delated.

Review

The design used within this project covers the required basis for the targeted specification. With minimal prior experience, there are a few notable areas from within the design that require addressing.

Due to the relatively minimal number of models required within a single sphere, no model list was used allowing a more direct interaction with every single model object, correctly lighting and shading each object with desired and specific techniques such as bump mapping, alpha blending… etc. Using a model list would prove to be more beneficial should the project be extended, with more model requirements the code management would become far harder to the point it would not even be efficient. However in the development of a model list it would be essential to take into consideration the different interactions with specific objects, especially lighting, which has to be calculated for each object; one of the issues observed would be a loop used to iterate through the model list in order to draw

each item has the potential to miscalculate lighting, even only apply lighting to the last drawn object which translates to the draw order at which point it becomes less efficient to go through the list and change the order of the index to one that works. If the assignment was redone, for longevity and potential of further development a model list design implementation would be a good feature, however for the current target specification can prove to be as equally inefficient to deal with if not more.

Another, design implementation that could have been done in a different way would be the use of pointer; the project overall would benefit greatly from the use of smart pointers. The use of smart pointers would remove the requirement of deleting pointer on shutdown thus removing the requirement of remembering to do so. Greatly decreasing the potential of memory leaks as well as being overall safer. Unfortunately, this was only something that was found during research later in the project and would not have been beneficial to be adapted and implemented at that point, however would likely be a focus for a remake.

One of the more notable issues that arose during development, was reading, and rendering a model; there are many available methods to achieve a model loader. Through research and overall a universally accepted file extension was decided to be ".obj", these files consist of several data segments:

- "V" lines contain the vertices
- "VT" lines contain the texture coordinates
- "VN" lines contain the normal vectors
- "F" lines contain each triangle face in the model

These data segments are great to read-in on a basic level, however during implementation a performance concern arose due to the pre-requirement of a vertex count; this means that the data is required to be read-in once to get an initial vertex count before the actual data can be processed, there is also a further potential issue which can be accounted for in regards to having to process each line individually, which increases the total amount of processing required. The solution, as suggested by several forum posts is to make a dedicated ".obj" parser, as this allows maximum efficiency potential as the parser can be catered to the desired result, only dealing with the required data as well as taking load off the actual program. A separate application was created to parse ".obj" files into dedicated ".txt" files only containing the desired data which is easily accessible to the model loader. Although this process is favoured, and would be a requirement in some form for bigger projects, due to the relatively small number of models used within this simulation, and no mesh animation expected, unfortunately there is no obvious benefit to the user.

3 Graphics

Algorithms

As outlined in the object parser in the “Review” of the previous section, my model objects are created based on a text file which comes from parsing in an “.obj” file.
The data from the “.obj” file is translated into the following format:

“vertices-x vertices-y vertices-z texture-x texture-y normal-x normal-y normal-z ”

Example:

```
12.6063 0.434289 4.18586 0.3494 0.6764 -1 0 -0
```

This format is grouped up 3 times to achieve a face. At the top of the page a “Vertex Count” is displayed allowing the creation of the model to be carried out with a single file read, achieving increased efficiency with less processing load.

```
for (int i = 0; i<m_vertex_count; i++)
{
        fin >> m_model[i].x >> m_model[i].y >> m_model[i].z;
        m_model[i].x *= scale;
        m_model[i].y *= scale;
        m_model[i].z *= scale;
        fin >> m_model[i].tu >> m_model[i].tv;
        fin >> m_model[i].nx >> m_model[i].ny >> m_model[i].nz;
}
```

Basic snippet of the loop used to read in vertex data.

The model class needs to re-iterate through the model structural data once to get all the required data to make the model object. From there the data can be used to calculate tangents and binormals using each 3 vertex lines as a face, this leads to the calculation of normals and thus the structural creation of the model object and its texture component.

The lighting pixel shader component is programmed to operate on each pixel to impact on rendering techniques such as bump mapping and specular, as used within the project.

```
texture_colour     = shaderTexture.Sample(SampleType, input.tex);
colour             = ambientColor;
specular           = float4(0.0f, 0.0f, 0.0f, 0.0f);
light_dir          = -light_direction;
light_intensity    = saturate(dot(input.normal, light_dir));

if (light_intensity > 0.0f)
{
colour             += (diffuseColor * light_intensity);
colour             = saturate(colour);
reflection         = normalize(2 * light_intensity * input.normal - light_dir);
specular           = pow(saturate(dot(reflection, input.viewDirection)), specularPower);
}

colour = saturate(colour);
colour = colour * texture_colour;
colour = saturate(colour + specular);
```

Code snippet from the light pixel shader.

The pixel shader samples each pixel colour from the texture using specific texture coordinate locations, based on what the vertex shader passes in. This process does not read the current screen, just works out what colour and transparency a pixel should be from the current primitive. These pixels then get put on the current draw buffer and displayed when processed.

Although, not implemented in the project the theory of adding basic shadows would be with the introduction of a shadow shader, and a depth shader. The depth shader would be responsible for sampling the generated scene to calculate depth distance based on the lighting, and the shadow shader will implement appropriate shadow shapes on the appropriate surfaces based what had been calculated. The reason behind the failed implementation of shadows within my project is potentially to do with the lighting, the depth calculations will include lighting positions. The shadow render will take in a texture, while the objects used within the project are all rendered with texture arrays, multiple textures to facilitate bump mapping and specular techniques. Overall, providing too many conflicts and the decision of developing shadows as a last feature was an oversight.

Bump mapping and specular render techniques used within the project are processed using a texture array combining up to 3 textural images for the specular technique and 2 for the bump mapping. The first texture image is the standard diffuse image, followed by a normal used to pronounce the elevated coordinates of the model, the third is specular which focuses more on the texture visible on the particular locations light makes contact.

The particle system used within the project takes in a single texture image to be used as the particle; it is processed with alpha blending to remove background and filled with a range of colours that can be defined.

```
while (i != index)
{
        m_particle_list[i].position_x       = m_particle_list[j].position_x;
        m_particle_list[i].position_y       = m_particle_list[j].position_y;
        m_particle_list[i].position_z       = m_particle_list[j].position_z;
        m_particle_list[i].red              = m_particle_list[j].red;
        m_particle_list[i].green            = m_particle_list[j].green;
        m_particle_list[i].blue             = m_particle_list[j].blue;
        m_particle_list[i].velocity         = m_particle_list[j].velocity;
        m_particle_list[i].active           = m_particle_list[j].active;
}
```

Code snippet from particle creation loop.

In the project the red and green range are set to 0 as only the blue range is desired for blue bubbles, this makes the bubble particles come out as a range of blue shades. The initial emitter position is defined while the individual bubbles are updated to go into a certain direction. Based on the frame time and the velocity set.

```
for (i = 0; i<m_current_particle_count; i++)
{
        m_particle_list[i].position_y =
        m_particle_list[i].position_y - (m_particle_list[i].velocity * frame_time);
}
```

Code snippet from particle update method.

Application-Objects and Graphics Representation

Each object file holds data on the textural coordination of each structural vertex and face, these data is passed within the object parser and is read within the model class, the compiled image textures are combined within an array of up to 3 images depending on the render technique, used within the render loop. It is within the render loop and the used shader technique that the lighting, transparency, texture calculations are carried out using the appropriate vertex and pixel shaders, to display the intended result.

Application-Object Behaviour Update

The render loop follows a simple process in which the appropriate raster states are enabled/disabled following the position/rotation changes of the upcoming object or object groups (should it be required) and rendered using the desired shader technique.

Potential Extensions

The project would overall suffer as scalability increases, although the model structural data process is improved by having a dedicated text format to be more efficient in processing the data, as mentioned in the previous section of “Review” the model class object will benefit from being adapted into a model list should the project scale up, in the current state it is arguably more efficient and easier to handle directly interacting with each object.

Adding basic shadows would be something that would take initial priority should the project be re-done as the choice in developing shadows as a final feature proved to be an oversight and was unable to function as intended. High-dynamic-range rendering would be a focus for an advanced feature, this allowing the preservation of details that may be lost due to limiting contrast ratios.

Additional Feature

Without any prior knowledge to the amount of work and time required to add this addition; actual mesh animations sound like a great implementation and addition to the project. Another interesting feature that doesn’t necessarily add anything to the project other than aesthetics is the implementation of a loading screen.

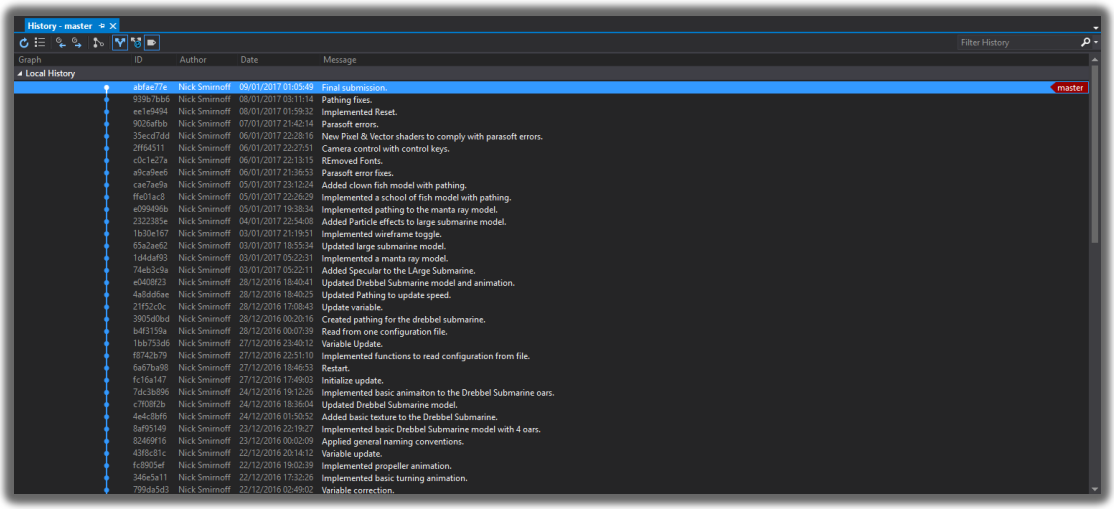
Overall, personally I would have been very excited and interested in working with Vulkan API, most likely made the project incredibly difficult with the technology being so new and not as much resources available to aid the development.

4 Project Management

Project management covered a few typical areas, mainly focusing on Git repository integration, professional practices and a minimalistic approach to agile programming done solo.

GitHub Integration

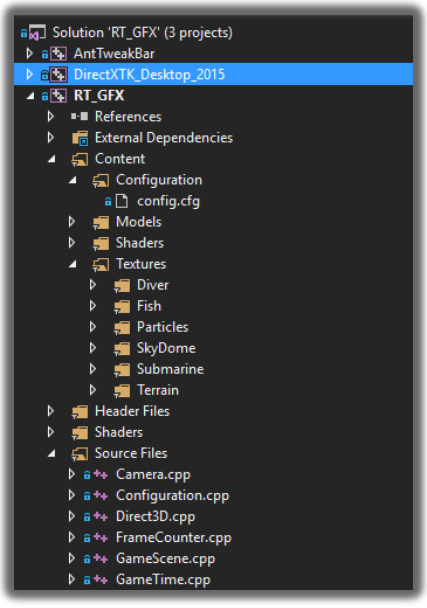
Project management mainly consisted within the integration of a web-based Git repository, GitHub. It offers all of the distributed version control and source code management functionality of Git, also with access to private repositories creates a layer of added security to the project and its contents. There are alternatives to achieve similar results such as SVN, however with the home integration, as well as the project insight tools available with GitHub, it proves to be a better choice.



Screenshot of my recent commit history to the dedicated GitHub repository within Visual Studio.

Visual Studio integration with GitHub made it simple to mange version control, allowing frequent commits to the server. Any additions or changes made are committed, with appropriate messages and the availability of reverting back to the previous working build should some errors arise gives the project an elevated form of efficiency when required to take such actions.

Professionalism



Screenshot of my folder/filter within Visual Studio.

Further management and professionalism efforts have been made in both the source folder structure as well as the source control within Visual Studio. Filters have been used to match the same folder structure as the source; maintaining a clear and efficient structure allowed a constant understanding of the content, making both file access and debugging procedures far more efficient.

Effort have been made and carried out to comply with typical coding practices such as regular commenting of code sections, whilst also referring with implementation of C++ style coding standards, which was done with reference with ‘Google C++ Style Guide’; mainly restricting camel-case. Furthermore, making appropriate additions/alterations to pass ‘Parasoft’ tests.

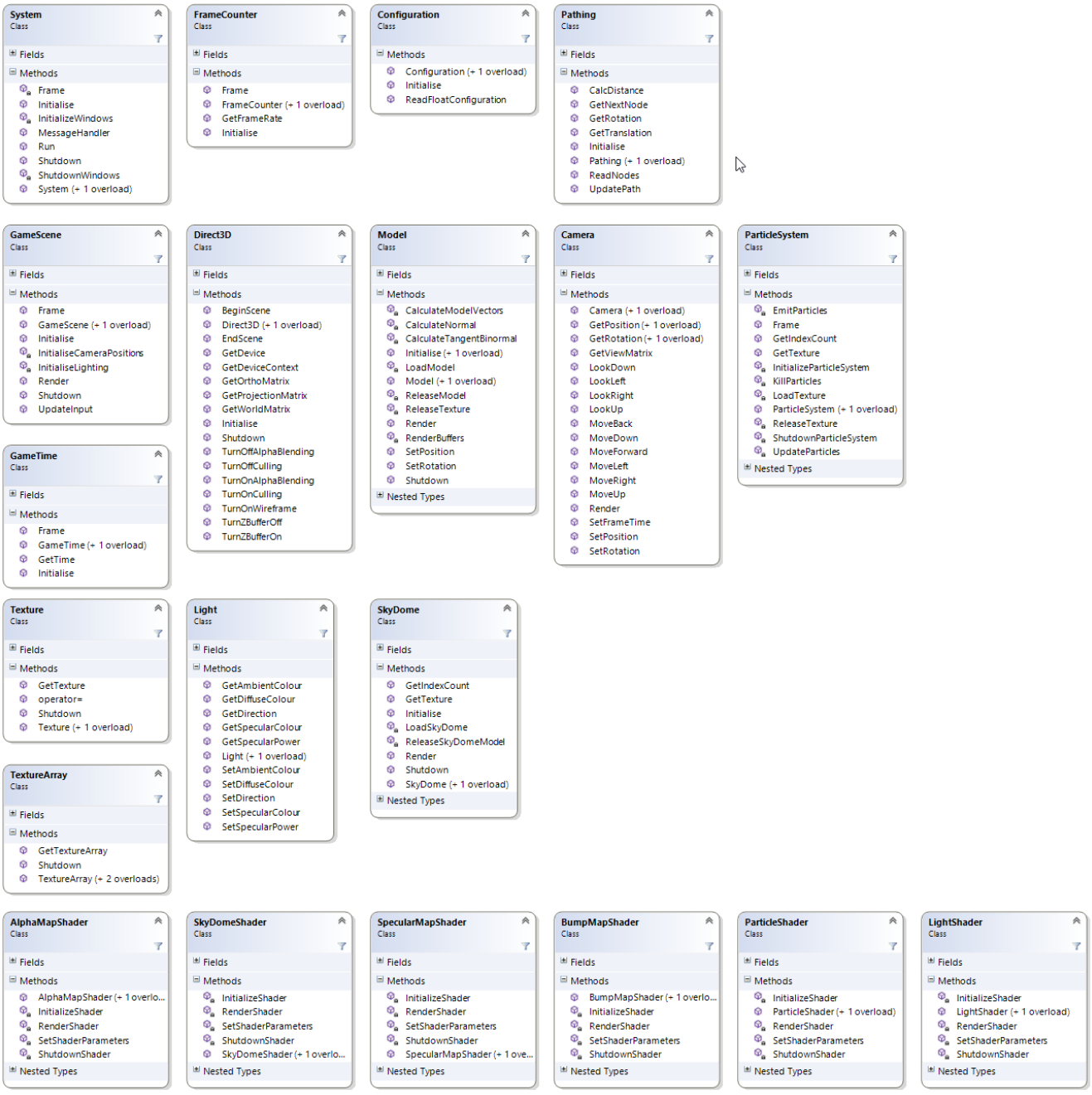
Planning

Planning and documentation of the project was accomplished, referencing the coursework specification. A project “README.md” file was created using a markdown language recognised by GitHub; this file outlined all the main topics regarding the project and expected functions and features. This file was regularly referred too at any point of development, especially during planning stages of what the next intended implementation would be. A solo form of agile ‘scrum’ programming was used; a task list was created in the form of the “README.md” file, from which the task would be selected and broken down to be implemented. Should the task require reference or a form of research, a website named ‘RasterTek’ would often be the place to start with research and implementation with any external or additional help coming from several forums and posts. This cycle would be repeated on completion or in one scenario where the implantation did not function correctly a previous build of the project was reverted to without any hassle, provided a fast and effective use of the GitHub integration.

5 Conclusion

In conclusion, and with consideration of the lack/no prior experience with the processes used to develop this project, the overall end project result turned out well. With the gathered experience, I am confident in my skill to produce a project that is of better quality.

Appendix A


System
Class
Fields
Methods
Frame
Initialise
InitializeWindows
MessageHandler
Run
Shutdown
ShutdownWindows
System (+ 1 overload)
FrameCounter
Class
Fields
Methods
Frame
FrameCounter (+ 1 overload)
GetFrameRate
Initialise
Configuration
Class
Methods
Configuration (+ 1 overload)
Initialise
ReadFloatConfiguration
Pathing
Class
Fields
Methods
CalcDistance
GetNextNode
GetRotation
GetTranslation
Initialise
Pathing (+ 1 overload)
ReadNodes
UpdatePath
GameScene
Class
Fields
Methods
Frame
GameScene (+ 1 overload)
Initialise
InitialiseCameraPositions
InitialiseLighting
Render
Shutdown
UpdateInput
Direct3D
Class
Fields
Methods
BeginScene
Direct3D (+ 1 overload)
EndScene
GetDevice
GetDeviceContext
GetOrthoMatrix
GetProjectionMatrix
GetWorldMatrix
Initialise
Shutdown
TurnOffAlphaBlending
TurnOffCulling
TurnOnAlphaBlending
TurnOnCulling
TurnOnWireframe
TurnZBufferOff
TurnZBufferOn
Model
Class
Fields
Methods
CalculateModelVectors
CalculateNormal
CalculateTangentBinormal
Initialise (+ 1 overload)
LoadModel
Model (+ 1 overload)
ReleaseModel
ReleaseTexture
Render
RenderBuffers
SetPosition
SetRotation
Shutdown
Nested Types
Camera
Class
Fields
Methods
Camera (+ 1 overload)
GetPosition (+ 1 overload)
GetRotation (+ 1 overload)
GetViewMatrix
LookDown
LookLeft
LookRight
LookUp
MoveBack
MoveDown
MoveForward
MoveLeft
MoveRight
MoveUp
Render
SetFrameTime
SetPosition
SetRotation
ParticleSystem
Class
Fields
Methods
EmitParticles
Frame
GetIndexCount
GetTexture
InitializeParticleSystem
KillParticles
LoadTexture
ParticleSystem (+ 1 overload)
ReleaseTexture
ShutdownParticleSystem
UpdateParticles
Nested Types
GameTime
Class
Fields
Methods
Frame
GameTime (+ 1 overload)
GetTime
Initialise
Texture
Class
Fields
Methods
GetTexture
operator=
Shutdown
Texture (+ 1 overload)
Light
Class
Fields
Methods
GetAmbientColour
GetDiffuseColour
GetDirection
GetSpecularColour
GetSpecularPower
Light (+ 1 overload)
SetAmbientColour
SetDiffuseColour
SetDirection
SetSpecularColour
SetSpecularPower
SkyDome
Class
Fields
Methods
GetIndexCount
GetTexture
Initialise
LoadSkyDome
ReleaseSkyDomeModel
Render
Shutdown
SkyDome (+ 1 overload)
Nested Types
TextureArray
Class
Fields
Methods
GetTextureArray
Shutdown
TextureArray (+ 2 overloads)
AlphaMapShader
Class
Fields
Methods
AlphaMapShader (+ 1 overlo...
InitializeShader
RenderShader
SetShaderParameters
ShutdownShader
Nested Types
SkyDomeShader
Class
Fields
Methods
InitializeShader
RenderShader
SetShaderParameters
ShutdownShader
SkyDomeShader (+ 1 overlo...
Nested Types
SpecularMapShader
Class
Fields
Methods
InitializeShader
RenderShader
SetShaderParameters
ShutdownShader
SpecularMapShader (+ 1 ove...
Nested Types
BumpMapShader
Class
Fields
Methods
BumpMapShader (+ 1 overlo...
InitializeShader
RenderShader
SetShaderParameters
ShutdownShader
Nested Types
ParticleShader
Class
Fields
Methods
InitializeShader
ParticleShader (+ 1 overload)
RenderShader
SetShaderParameters
ShutdownShader
Nested Types
LightShader
Class
Fields
Methods
InitializeShader
LightShader (+ 1 overload)
RenderShader
SetShaderParameters
ShutdownShader
Nested Types

References

GitHub, I., 2016. *GitHub.* [Online]
Available at: https://github.com/
[Accessed 24 November 2016].
Google, 2016. *Google C++ Style Guide.* [Online]
Available at: https://google.github.io/styleguide/cppguide.html
[Accessed 22 December 2016].
NA, 2016. *Tutorial Index.* [Online]
Available at: http://www.rastertek.com/tutindex.html
[Accessed 18 December 2016].