**C++ Programming and Design and Real-Time Graphics**

**Software Portfolio Report**

Submitted for the MSc in

Advanced Computer Science

January 18

By

**Alexander C Whitehead**

Word Count: 2701

Table of Contents

[1 Design 5](#_Toc503510812)

[1.1 Initial Class Diagram 5](#_Toc503510813)

[1.2 Main Class Diagram 6](#_Toc503510814)

[1.3 Utility Class Diagram 7](#_Toc503510815)

[1.4 Class Descriptions 8](#_Toc503510816)

[1.4.1 CPU 8](#_Toc503510817)

[1.4.2 Direct3D 8](#_Toc503510818)

[1.4.3 Face 8](#_Toc503510819)

[1.4.4 FPS 8](#_Toc503510820)

[1.4.5 Graphics 8](#_Toc503510821)

[1.4.6 Graphics Loader 8](#_Toc503510822)

[1.4.7 Image 8](#_Toc503510823)

[1.4.8 Image Shader 9](#_Toc503510824)

[1.4.9 Image Vertex 9](#_Toc503510825)

[1.4.10 Keyboard Events 9](#_Toc503510826)

[1.4.11 Light 9](#_Toc503510827)

[1.4.12 Model 9](#_Toc503510828)

[1.4.13 Model Loader 9](#_Toc503510829)

[1.4.14 Model Shader 9](#_Toc503510830)

[1.4.15 Model Vertex 9](#_Toc503510831)

[1.4.16 Particle 10](#_Toc503510832)

[1.4.17 Particles 10](#_Toc503510833)

[1.4.18 Particles Shader 10](#_Toc503510834)

[1.4.19 Particle Vertex 10](#_Toc503510835)

[1.4.20 Scene 10](#_Toc503510836)

[1.4.21 Shadow Shader 10](#_Toc503510837)

[1.4.22 Text 10](#_Toc503510838)

[1.4.23 Text Loader 11](#_Toc503510839)

[1.4.24 Texture Array 11](#_Toc503510840)

[1.4.25 TGA 11](#_Toc503510841)

[1.4.26 Timer 11](#_Toc503510842)

[1.4.27 View 11](#_Toc503510843)

[1.5 Interaction Diagram 12](#_Toc503510844)

[1.6 Design Critique 13](#_Toc503510845)

[1.6.1 Advantages 13](#_Toc503510846)

[1.6.2 Disadvantages 13](#_Toc503510847)

[1.6.3 Changes from Initial Design 13](#_Toc503510848)

[1.6.4 Possible Future Changes 13](#_Toc503510849)

[2 Graphics 14](#_Toc503510850)

[2.1 Algorithms 14](#_Toc503510851)

[2.1.1 Geometry Representation 14](#_Toc503510852)

[2.1.2 Lighting 14](#_Toc503510853)

[2.1.3 Shadows 14](#_Toc503510854)

[2.1.4 Particle System 15](#_Toc503510855)

[2.2 Rendering 16](#_Toc503510856)

[2.3 Updating Objects 17](#_Toc503510857)

[2.4 Extensions and Scalability 18](#_Toc503510858)

[2.5 Additional Features 19](#_Toc503510859)

[3 Project Management 20](#_Toc503510860)

[3.1.1 Processes and Methodology 20](#_Toc503510861)

[3.1.2 Source Control 22](#_Toc503510862)

Table of Figures

[Figure 1: This image shows the initial UML Class Diagram for the project. 5](file:///Z:\Real-Time%20Graphics\Software%20Portfolio%20Report%20and%20Video\Software%20Portfolio%20Report.docx#_Toc503510863)

[Figure 2: This image shows the final UML Class Diagram for the main classes of the project. 6](file:///Z:\Real-Time%20Graphics\Software%20Portfolio%20Report%20and%20Video\Software%20Portfolio%20Report.docx#_Toc503510864)

[Figure 3: This image shows the final UML Class Diagram for the utility classes of the project. 7](file:///Z:\Real-Time%20Graphics\Software%20Portfolio%20Report%20and%20Video\Software%20Portfolio%20Report.docx#_Toc503510865)

[Figure 4: This image represents the interactions/messages passed between classes during the updating and then rendering of a model. The arrows represent a message passed from one class to another, the comments show the message and the order of operations is from top left to bottom right. 12](file:///Z:\Real-Time%20Graphics\Software%20Portfolio%20Report%20and%20Video\Software%20Portfolio%20Report.docx#_Toc503510866)

[Figure 5: This image shows a graphical representation of three of the most popular project management methodologies. 20](file:///Z:\Real-Time%20Graphics\Software%20Portfolio%20Report%20and%20Video\Software%20Portfolio%20Report.docx#_Toc503510867)

[Figure 6: This image shows a graphical representation of the Iterative and Incremental software development methodology. 21](file:///Z:\Real-Time%20Graphics\Software%20Portfolio%20Report%20and%20Video\Software%20Portfolio%20Report.docx#_Toc503510868)

[Figure 7: This image shows the commit history for the repository that the project was stored in. 22](file:///Z:\Real-Time%20Graphics\Software%20Portfolio%20Report%20and%20Video\Software%20Portfolio%20Report.docx#_Toc503510869)

# Design

## Initial Class Diagram

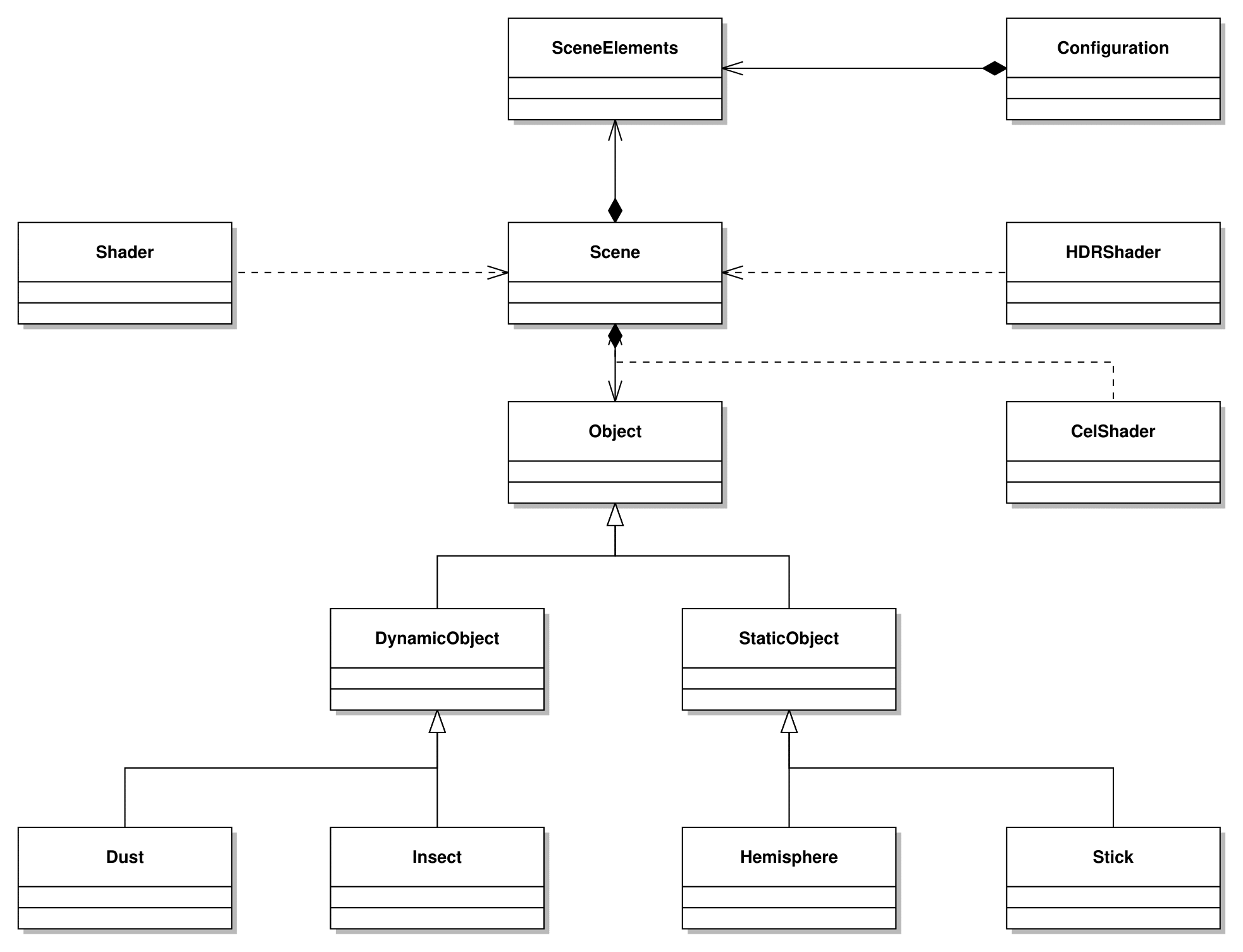


Figure : This image shows the initial UML Class Diagram for the project.

## Main Class Diagram

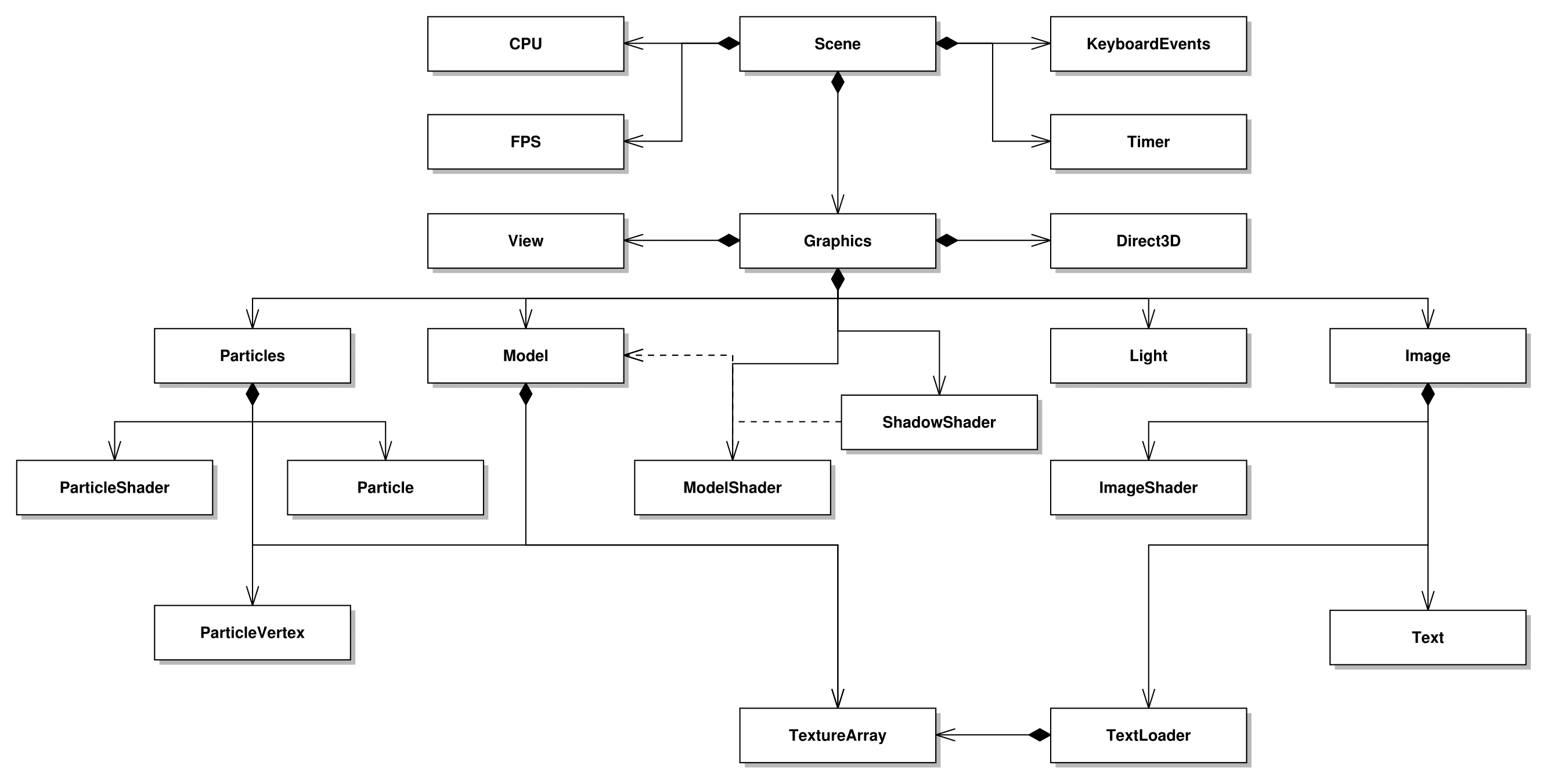


Figure : This image shows the final UML Class Diagram for the main classes of the project.

## Utility Class Diagram

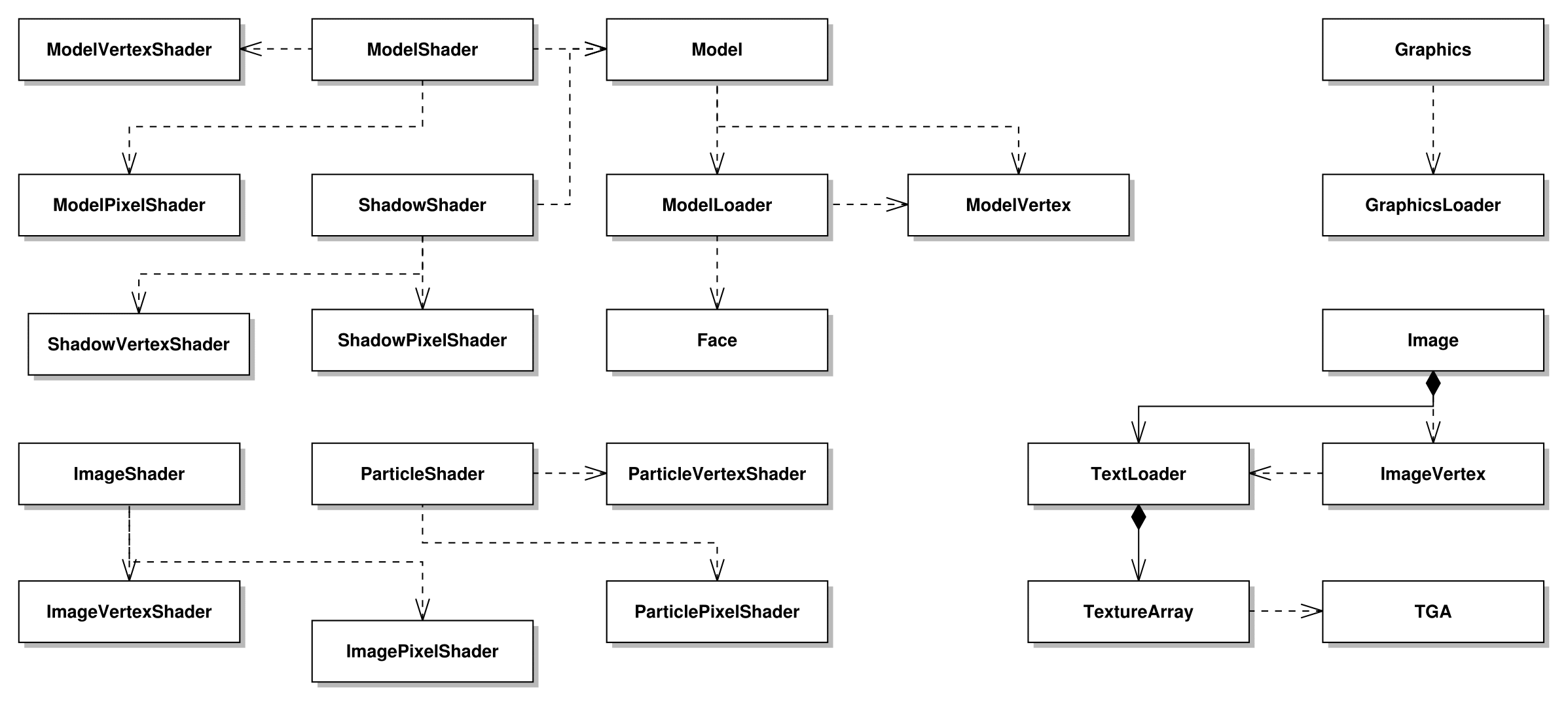


Figure : This image shows the final UML Class Diagram for the utility classes of the project.

## Class Descriptions

## CPU

The CPU class is responsible for updating a variable which represents the current workload of the CPU as a percentage.

## Direct3D

The Direct3D class is responsible for all of the variables associated with the Direct3D framework, beginning a scene by clearing it to a set colour and ending a scene by switching the two buffers which are being rendered to by the program.

## Face

The Face class is a container class that is responsible for the vectors of one face of the model plus the faces texture coordinates and normal values.

## FPS

The FPS class is responsible for updating a variable which represents the current framerate of the application in frames per second.

## Graphics

The Graphics class is the main class associated with rendering to the scene and as such is responsible for holding and updating every graphics related object in the program and then rendering these images to the scene.

## Graphics Loader

The Graphics Loader class is responsible for initialising a Graphics object with values held within a configuration file.

## Image

The Image class is responsible for variables which are to be rendered to the scene as an image. The Image class is also responsible for updating the content of these images.

## Image Shader

The Image Shader class is responsible for compiling the Image Pixel Shader and the Image Vertex Shader and then loading the relevant data into these shaders to render a given image to the screen.

## Image Vertex

The Image Vertex class is a container class that is responsible for holding the position and texture coordinates of one vertex of an image object.

## Keyboard Events

The Keyboard Events class is responsible for updating an array which represents the current state of each key on the keyboard.

## Light

The Light class is responsible for updating variables which represent the current position of a light in the scene, its colour, velocity and the path that the light follows.

## Model

The Model class is responsible for holding variables which represent a given model which is to be rendered to the scene. The Model class is also responsible for updating the position and rotation of these models and for rendering these models to the scene as either a shadow or a model.

## Model Loader

The Model Loader class is responsible for loading an OBJ model files into a model object at a given unit scale.

## Model Shader

The Model Shader class is very similar to the Image Shader class. However, it deals with models rather than images.

## Model Vertex

The Model Vertex class is very similar to the Image Vertex class. However, it also contains normal, tangent and binormal values.

## Particle

The Particle class is a container class that is responsible for holding the position and velocity of one particle.

## Particles

The Particles class is responsible for holding a collection of Particle objects which are to be rendered to the scene. The Particles class is also responsible for updating the position and rotation of these particles and adding and removing Particle objects whenever a given condition is met and for rendering these particle objects to the scene.

## Particles Shader

The Particles Shader class is very similar to the Model Shader class. However, it deals with particles rather than models.

## Particle Vertex

The Particle Vertex class is very similar to the Image Vertex class.

## Scene

The Scene class is the main class for the program and as such is responsible for holding and updating every other object in the program. The Scene class is responsible for dealing with message callbacks to and from the operating system and it is responsible for dealing with input from the user.

## Shadow Shader

The Shadow Shader class is very similar to the Particles Shader class. However, it deals with rendering models as shadows rather than particles.

## Text

The Text class is responsible for holding variables which represent a string of text which is to be rendered to the scene as an image. The Text class is also responsible for updating the content, position and colour of this text for rendering this text to the scene as an image.

## Text Loader

The Text Loader class is responsible for holding the character positions of the text texture.

## Texture Array

The Texture Array class is responsible for loading and storing the TGA textures required for one model.

## TGA

The TGA class is a container class that is responsible for holding the variables associated with decoding TGA texture files.

## Timer

The Timer class is responsible for updating a variable which represents the difference in time between when each frame of the scene is rendered.

## View

The View class is responsible for holding and updating variables which represent the positions and rotations of each camera. The View class is responsible for generating a view matrix from the current camera position and rotation.

## Interaction Diagram

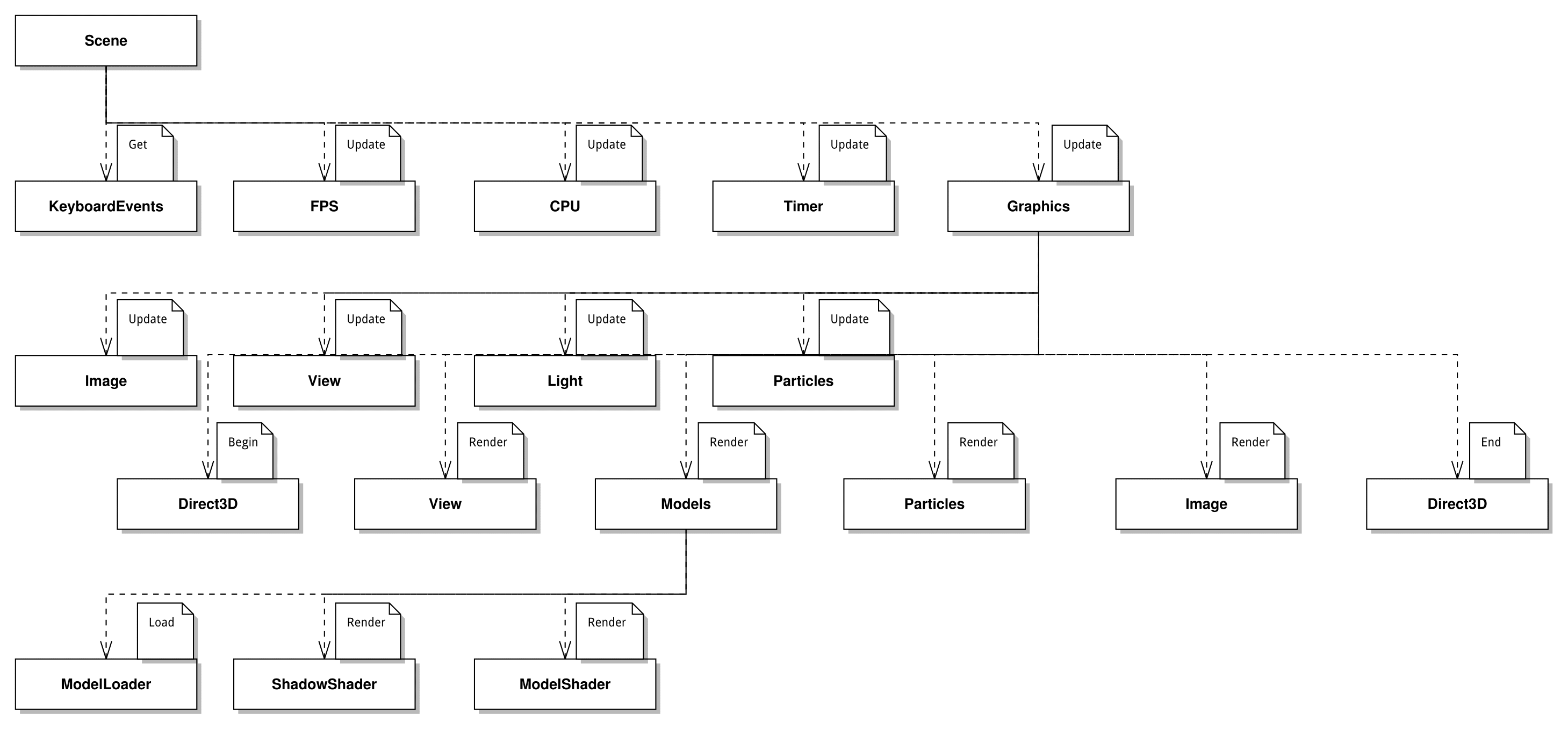


Figure : This image represents the interactions/messages passed between classes during the updating and then rendering of a model. The arrows represent a message passed from one class to another, the comments show the message and the order of operations is from top left to bottom right.

## Design Critique

## Advantages

One of the main advantages of the design is the fact that it has been broken down into its smallest component parts, thus a lot of the implementation is abstracted away. This means that the code is quite portable and it is quite easy for another developer to come along and implement new features in the program without it interfering with previously implemented features.

Because of the application of abstract data in the code it is also not necessary for a developer to understand exactly how the program is implementing a function, all which is required is that the developer understands what the aim of a given function is to use it. This is similar to the way in which an API would be understood by a developer e.g. Direct3D.

## Disadvantages

One disadvantage of the program is that there is a lack of the use of interfaces and as such there are instances of similar code being repeated in multiple places rather than one abstract class being written.

An example of where this would have been useful would be that each object in the program is fully defined separately from every other object, it would have been advantageous to have an object class that each type of object inherited from.

## Changes from Initial Design

The main changes from the initial design were that high dynamic range shaders and cel shaders were never implemented in the final program and composition was favoured over inheritance, also obviously numerous utility classes were added that were not considered during the initial design phases.

These differences are highlighted when comparing the initial and final UML class diagrams above.

## Possible Future Changes

In hindsight if the project was to be done again a few changes which should be incorporated would include; using interfaces, virtual functions and abstract classes, using DirectTK to load models rather than writing a proprietary OBJ parser and conforming to standard C++ practices rather than for instance using C style arrays.

# Graphics

## Algorithms

## Geometry Representation

To load geometry onto the graphics card to be rendered the program first parses a model, into a proprietary file format. This file format represents each face of the model as 3 vertices followed by texture coordinates and then a vector representing the normal direction of the face. This file is then read sequentially into memory.

In order to then render these faces they are loaded onto the graphics pipeline for the relevant shader to render as a list of individual triangles.

This is advantageous as it is easier to program and less likely for problems to occur even on fringe condition.

However, it requires more resources without a gain in graphical fidelity.

## Lighting

To simulate lighting the Blinn-Phong lighting model is used. The Blinn-Phong lighting model is made up of 3 individual lighting components which are summed together to achieve the overall effect, firstly a global ambient lighting is applied to the scene, this is used as the baseline lighting level, this ambient light is only ever really apparent when no other light can reach an object.

The second kind of light is diffused light, this light represents light which is reflected off of a ‘rough’ surface. The intensity of this light is determined by the normal direction relative to the light source.

The third kind of light is specular light, this light represents light reflected off of a ‘smooth’ surface into the ‘eye’ of the observer, and this kind of light appears as bright highlights on the objects surface. The intensity of this light is determined by the angle between the normal, the light source and the ‘eye’ of the observer and the reflectivity of the surface.

The main advantages of this lighting model is that it uses resources sparingly while still delivering a visually pleasing experience.

A disadvantage of this lighting model is that the simulation of lighting is not realistically physically based.

## Shadows

The planar shadow technique was used to generate shadows. This works by generating a world matrix which collapses a given model into the shadow plane from the perspective of a light source.

The main advantage of this method is that it is incredibly easy to implement and does not require such techniques as rendering to a texture.

However, this form of shadowing is incredibly unrealistic as the shadows cannot be cast onto other objects.

## Particle System

To implement a particle system a class was created which holds a vector of particles, these particles track their own position. After a set time passes a new particle is added to the vector with a semi-random position, based upon the current position of the dragonfly object. When a particle drops below the ground plane of the scene it is removed from the vector. Each particle is a single triangular face which is alpha blended, these objects are billboarded with relation to the ‘eye’ of the observer to ensure that each particle always faces the observer. On each update the velocity of each particle is updated with regards to gravity using Euler integration.

The main advantage of this algorithm is that it uses resources sparingly and also achieves a realistic simulation using Euler integration.

A disadvantage of this implementation is that because of the particular textures and alpha blending methods used the particle system graphically appears unrealistic.

## Rendering

Application objects are connected to their graphical representations using a buffer which contains the vertices of the object used to render the object, when an object is rendered the world matrix used to render the object is translated and rotated by vectors which are contained within the object, the world matrix is reset after each object is rendered.

The advantage of this approach is that is a very simple human readable approach and uses the least memory out of any viable approach.

A disadvantage of this approach is that it is possible while using Euler angles for two of the axis of rotation to become aligned and for changes in either of these two axis to apply the same rotation to the object, this is known as gimbal lock.

## Updating Objects

When the state of the program changes this information is propagated throughout the program using signalling variables which all objects have pointers to, for instance when the keyboard event occurs which indicates that the dragonfly should start flying a variable is incremented to indicate that the dragonfly should begin its flying animation, once the dragonfly is at the required altitude the variable is incremented again.

This variable is also passed into the vertex shader which uses it to determine if the wings or legs of the dragonfly should be moving and in which direction.

## Extensions and Scalability

One extension which could be included in the project would be shadow mapping and volumetric shadows. Currently a shadow can only be projected onto the shadow plane

However, the implementation of shadow mapping and volumetric lighting would mean the inclusion of a render to texture method. This is because shadow mapping works by rendering the scene from the perspective of a light source to a texture which can then be used later in the model shader to determine if a light source can see an object and therefore if it can cast light onto it.

The implementation of this functionality would have pushed this project overtime.

This project is relatively scalable, this is because most of the objects in the program are dynamically allocated.

However, one issue with the scalability of this project is that deferred rendering was not implemented. One use of deferred rendering is to ensure that only things that can be seen by the ‘eye’ of the observer are rendered to the scene, this is advantageous as no unnecessary rendering occurs.

Because deferred rendering was not implemented scenes would have to be small so as to ensure the programs performance is no hindered.

## Additional Features

One feature which would improve the project would be screen space ambient occlusion. This is the use of ambient lighting maps to ensure that unlit objects do not appear to be lit with the same intensity. this occurs in the the current lighting model because no matter how concealed an object is it will still be lit to the same intensity, for instance using the current lighting model an unlit underground bunker would still be appear to be lit with ambient lighting.

However, to implement this feature a lighting method other than the Blinn-Phong Lighting model would have to be implemented as the Blinn-Phong lighting model is inherently incompatible with screen space ambient occlusion.

# Project Management

## Processes and Methodology

When starting any kind of large project, it is always a good idea to follow a project management methodology. Proper project management can highlight issues with the design of the project before they become too much of an issue or hamper the overall performance or delivery time of the project.

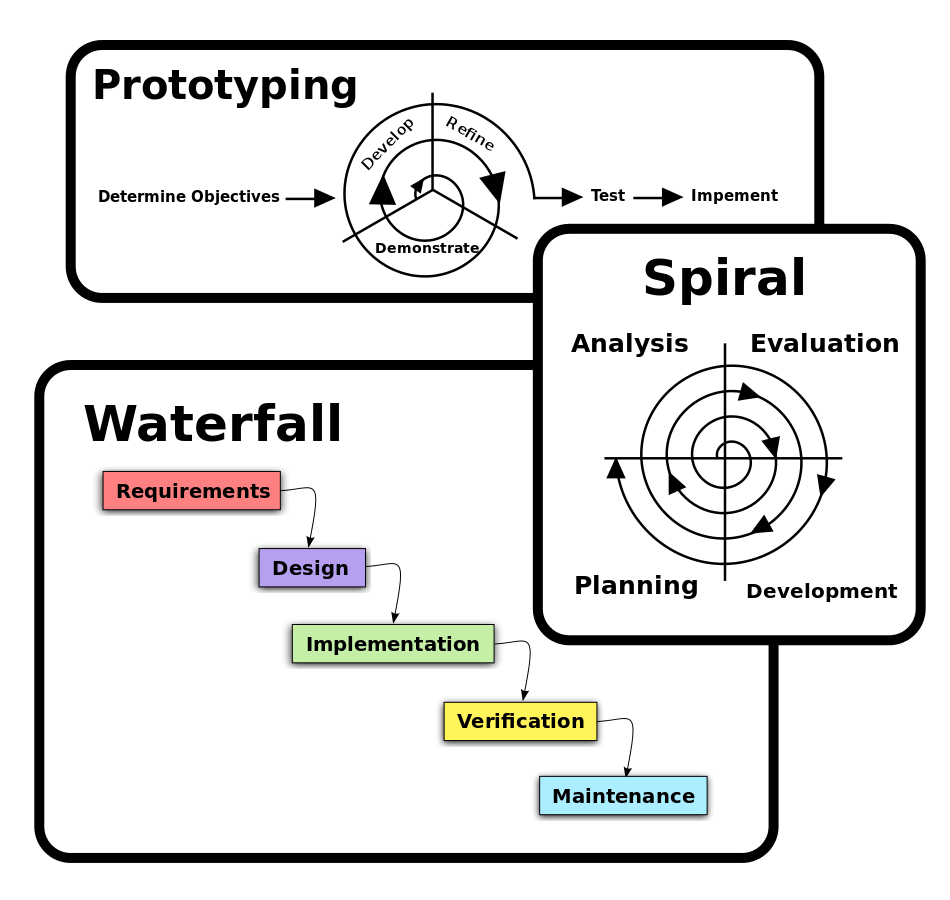


Figure : This image shows a graphical representation of three of the most popular project management methodologies.

By far and away the Waterfall methodology is the most popular project management methodology, this is because the Waterfall methodology is the oldest and most widely known form of project management.

The Waterfall methodology is rigid and linear, each task must be completed before the next task can begin, and tasks flow like a waterfall, hence the name.

The main advantage of the Waterfall methodology is that it is widely understood by people unfamiliar to software development, this is because the Waterfall methodology can be applied to any kind of project, not necessarily just software development projects. Thus it is possible for a manager with no prior knowledge of software development to manage a software development project.

However, in practice the Waterfall methodology is not ideal as it does not consider the inevitable scope creep apparent with most software development projects, this is because the Waterfall methodology does not allow for changes to be dynamically made to the steps of the project. Thus most software development projects managed using the Waterfall methodology run overtime.

In this instance, the Iterative and Incremental methodology was followed.

This software development methodology addresses most of the issues apparent in the Waterfall methodology, for example, rather than being rigid and linear the Iterative and Incremental methodology allows for a certain amount of flexibility by setting certain goals which are delegated out in cycles to be completed in a set timeframe converging on a deadline whereby a new piece of functionality will have been implemented.

Also, because of the continuous testing apparent in the cycles of the Iterative and Incremental methodology it is more likely that issues are identified and dealt with earlier in the timeframe of the project.

One disadvantage of the Iterative and Incremental methodology is that each task is undertaken entirely by one developer or group of developers, thus each developer needs to be trained to a similarly high standard. However, because this project was undertaken by a sole developer this issue is irrelevant.

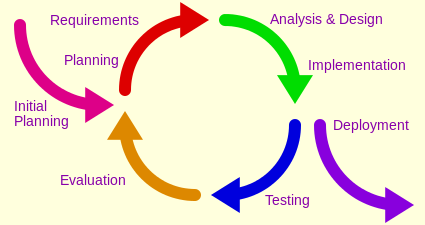


Figure : This image shows a graphical representation of the Iterative and Incremental software development methodology.

To save time and resources as well as to make the program more understandable the project was written to be self-documenting.

## Source Control

When starting any kind of large project, it is always a good idea to have some form of a backup or source control.

Source control allows a large group of people to collaborate on a project and merge their joint efforts together.

Source control is also used in case of the loss of work either through the local copy of the work being lost or destroyed or because a change has been made which needs to be removed. Source control tracks the changes between different versions of the same project and allows the users to reacquire any previous version of the project.

For this project Subversion was chosen as the main form of source control, this is because the University of Hull offers its own Subversion server. Although versions of the project were also stored on GitHub.

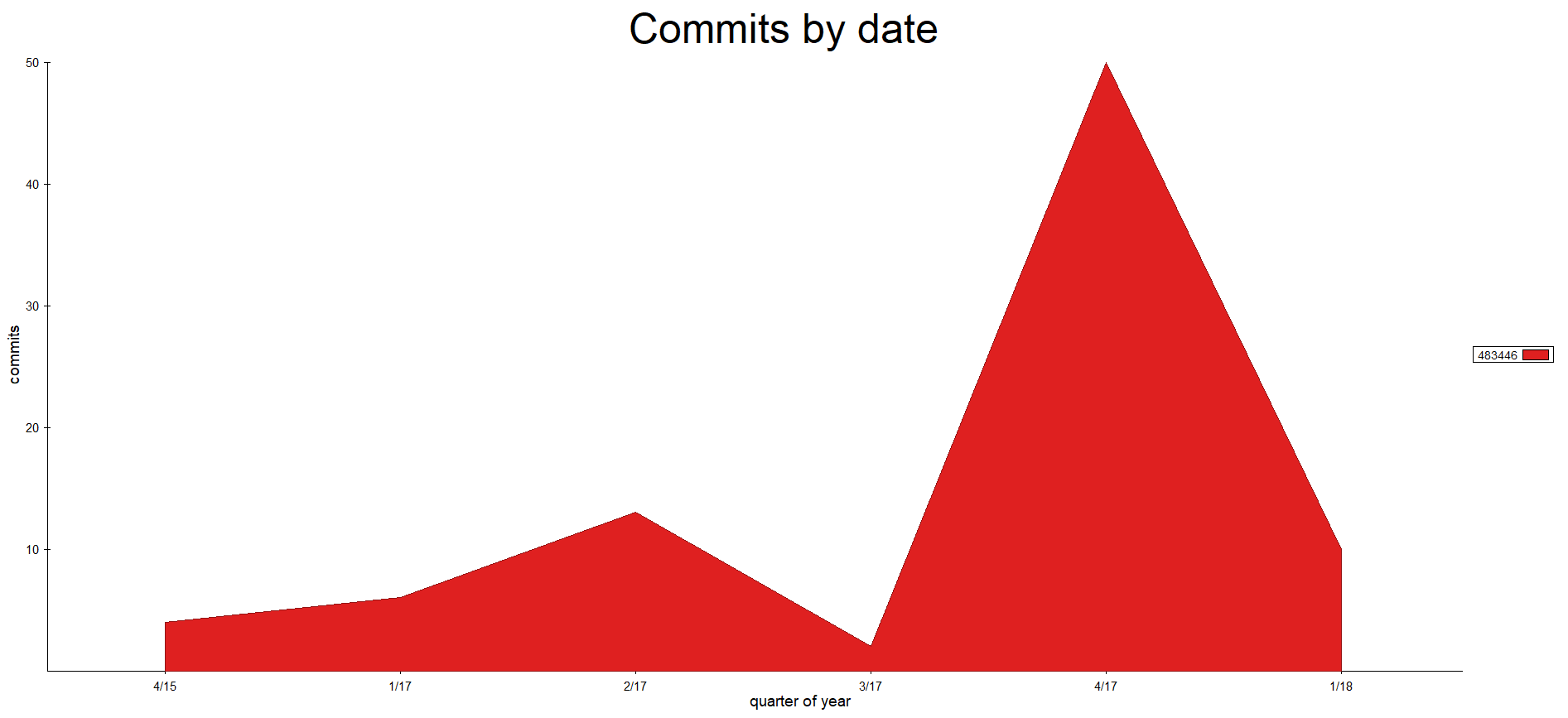


Figure : This image shows the commit history for the repository that the project was stored in.