Tomáš Zemanovič
Matric No: S1033008

Suitability of Functional Programming Concepts for 3D Game Engine Development in C++

BSc (Hons) Computer Games (Software Development)
2015

Project Supervisor: Brian McDonald BSc PgD

Second Marker: Robert Law

Submitted for the Degree of BSc in
Computer Games (Software Development), 2014 – 2015

"Except where explicitly stated all work in this report including the appendices, is my own"

Signed: _____ **Date:** _____

Acknowledgement

I would like to express deep appreciation to my loving partner Kayleigh, family, Jaebles and friends, for their support during studies, my project supervisor Brian McDonald for accepting this project's topic proposal and his guidance throughout the development and Microsoft DreamSpark for providing free academic licenses to their software.

Contents

Abstract

The aim of functional programming is to provide higher level abstraction further away from hardware than imperative programming, which promises benefits on numerous fronts. It has already been successfully integrated into many areas of software development however in games development its adoption has been mostly only theoretical. Arguably, one of the major reasons is that computer games, being soft real-time applications, cannot afford the performance loss, from which functional programming languages tend to suffer. However, as the code complexity is growing even further with multi-core programming, the threshold under which it is feasible to trade performance for productivity increases.

To an extent, it is possible to use functional programming concepts in C++, the most popular language for games development, which has been predicted to keep its position, even further with all the new features that are being implemented with every new standard. Many ideas that are being brought into C++ gain inspiration from functional programming, which should make its adoption in C++ simpler yet.

The goal of this project was to investigate how practical it is to combine functional programming principles with traditionally imperative language, in development of 3D game engines. By using develop and test methodology, a limited 3D game engine is implemented in a subset of C++ following selected functional programming principles, together with an imperative implementation with equal capabilities for comparison and evaluated by development of demonstration applications with identical specifications for both versions.

The expected result was a more concise, reliable and modular code, but also possibly less efficient and due to C++ syntax. more difficult to read, which was mostly confirmed by the findings from project's evaluation. The biggest issues found to be holding functional programming back in C++ is increase in code verbosity, which harms productivity, code's readability and maintainability, and decrease in efficiency of the game engine, caused mostly by under-developed optimisation techniques used by VC++ compiler for functional programming constructs such as lambda and closure functions.

The findings of this project have been envisaged to contribute to knowledge about further development of functional programming paradigm in C++ and open the dialog for the possible direction of game engine development in future.

1 Introduction

This chapter is intended to help to familiarize the reader with challenges that current game engine development practices have to face and to find out about possible solutions through adoption of functional programming concepts, which is ultimately the aim of this project. The last section of this chapter outlines the structure of the rest of the report.

1.1 Project Background

The aim of functional programming is to provide higher level abstraction further away from hardware than imperative programming, which promises benefits on numerous fronts. It has already been successfully integrated into many areas of software development however in games development its adoption has been mostly only theoretical. The selection of games written in functional programming style is limited, some of these are analysed in section 2.3.1.2. Arguably, one of the major reasons is that computer games, being soft real-time applications, cannot afford the performance loss, from which functional programming languages tend to suffer. However, as the code complexity is growing even further with multi-core programming, the threshold under which it is feasible to trade performance for productivity increases (Carmack, 2012).

C++ is the most popular language for computer games engine development and it has been predicted to its position even further with all the new features that are being implemented with every new standard. To an extent, it is possible to use functional programming concepts in C++. In fact, many ideas that are being brought into C++ gain inspiration from functional programming, which should make its adoption in C++ simpler.

The first higher level functional language Lisp was invented only a year after FORTRAN made its appearance and since then has spawned many different dialects. Other widely used functional languages are ML (or its better known dialects OCaml and Standard ML), Erlang and Haskell and from the more recent ones Scala, Clojure and F#. The functional programming (from now on FP in short) concepts considered in this project will be mainly deduced from Haskell. Haskell is a purely functional language, which forces its users to "think functionally" (Callaghan, 2012), without being able to fall back to imperative style. It is a leading pure functional language in research, but it has been adopted in many real-world projects as well by companies such as Barclays Capital, Facebook and Google, with reportedly great success (Marlow, et al., 2014; Pop, 2010; Frankau, et al., 2009).

Programming in imperative languages can be thought of as stating *how* to achieve something (series of steps) as opposed to FP languages in which the ideas of *what* programs should do are expressed (the logic of computation is declared without its control) (Lloyd, 1994). In that sense FP is higher level than imperative programming, as it changes focus of its users on higher level logic rather than implementation details. A quick high-level overview of FP concepts follows.

Meijer (2014) has demonstrated that the same way the functional programming paradigm can be followed in traditionally imperative languages it is also possible to write imperative code in functional languages like Haskell, which is often called the world's best imperative language, because of the safety of code that is checked at compile time rather than at runtime, where errors can result in program crashes for many different reasons. These can be multithreading issues, memory & time leaks or general programming errors.

1.1.1 Functional Programming Concepts

1.1.1.1 First-class functions

At the core of FP are first-class functions. When a function is first-class citizen, it can be operated on as if it was data – it can be stored in data structures, created anonymously at runtime, passed as an argument to other functions and returned from functions (Backfield, 2014). First-class functions also typically obey rules of lexical scoping just like variables do.

1.1.1.2 Higher-order Functions

Higher-order functions, as a direct result of first-class functions, are the means of composing functions with other functions to solve problems. Higher-order function takes a function as an argument or returns a function or does combination of both (Hudak, 2000). Higher-order functions are the essence of FP.

1.1.1.3 Currying

Currying is a technique of breaking down functions that require multiple arguments into series of functions where each function takes a single argument while keeping its functionality untouched (Lipovača, 2011). This technique allows a partial application of functions – defining a new function from function that takes multiple arguments by passing it less than required number of arguments, which can promote reuse of more abstract functions.

1.1.1.4 Immutable Data

Mutating data introduces side effects which can be a root of many bugs, the reason being it can be difficult to keep track of all the possible side effects of a function and understand all the possible states (Backfield, 2014). This difficulty grows further in the world of multithreaded and parallel processing code that faces data races, dead locks and other subtleties that can be extremely difficult to debug and fragile to slightest changes (Williams, 2012). FP avoids these kinds of problems with philosophy "you cannot break what you cannot change".

1.1.1.5 Purity

Pure functions always evaluate the same output for the same set of input values (this property, also called Referential Transparency, means that anywhere where a variable is in scope it has the same value) and they do not modify any outer state and thus produce side effects (O'Sullivan, et al., 2008). Naturally, some functions do have to modify the program state such as functions producing output to I/O devices. One of the ways to deal with this issue in FP is to minimalize these impure functions as much as possible and separate them from the rest of the program. Languages that follow this rule are called pure functional languages (like Haskell).

1.1.1.6 Algebraic and Recursive Data Types

Haskell gets a lot of praise for its well-designed powerful algebraic strong data type system. It promotes reuse through composition of primitive data types in combination with product and sum operations together with unit type (Chakravarty, 2008).

Recursive data types can contain values of the same type as their owner. They can grow dynamically and are well suited for structures such as lists and trees (Okasaki,

1998). Typically, languages support pattern matching alongside algebraic and recursive data types, which can make operations on such structures much more comfortable.

1.1.1.7 Recursive Functions

Not being able to change state means one cannot simply use loops (`for`, `while`, `do while`, etc.), but that it is not a problem because such constructs can be replaced with recursive function, which is a function that calls itself inside its body definition (Lipovača, 2011).

Using recursion instead of loops can be beneficial because many mathematical functions are defined using recursion too, so they can be easily used without having to be transformed into loops first, which is not always a trivial task, as can be seen in the text Functional C (Hartel & Henk, 1999), which teaches techniques on how to achieve this.

If recursion is not optimized it can easily lead to stack overflow, that is why functional languages and even some imperative languages implement tail-recursive call optimization. Tail-call recursion means that the call of recursive function is the last action that happens within this function scope. This allows for optimization because the compiler knows that it can eliminate the stack frame for each iteration after current iteration has been executed because it is no longer needed.

1.1.1.8 Lazy Evaluation

In pure functional code the order in which functions are evaluated does not change their functionality and therefore they can use technique called lazy evaluation. Lazy evaluation delays computation of an expression until it is absolutely necessary or requested, thus it can avoid doing work that is not needed which can improve efficiency of certain algorithms and allows usage of infinite data structures that can increase expressivity of a language (Mena, 2014).

1.1.1.9 Lambda expressions & Closures

Lambda expressions (also known as anonymous functions or function literals) are unnamed functions that help to write more concise code in cases where such functions are only used once and are commonly used in conjunction with higher-order functions (Lipovača, 2011).

Closures are closed over lambda expressions that capture value or store reference to non-local variables (also called free variables) without the necessity of passing these variables into the function as arguments. Such functions can subsequently access these variables even when they are called outside of their lexical scope (Backfield, 2014).

1.1.2 Functional Programming Adoption

The FP concepts listed above bring with them many benefits, in the publications and experiments that discuss this topic the most commonly mentioned ones being (Mena, 2014; Lipovača, 2011; O'Sullivan, et al., 2008; Hudak & Jones, 1994; Hughes, 1990; Backus, 1978):

- Increased productivity
- Better modularity and extensibility
- Better testability and reliability

- Simplified concurrency
- Rapid prototyping

However, it is not all as positive, as according to research done by InfoQ (Schuster, 2012) FP languages are held back by these issues (presented in order of the poll results, starting from the most critical ones):

- Lack of developers trained in functional programming
- Insufficient tool support
- Functional features already exist in mainstream imperative languages
- Lack of libraries
- Use of academic terminology
- Poor interoperability with existing codebases
- Unclear or missing vendor support
- Hard to predict performance behaviour

The last problem is arguably the most critical for computer games development. FP languages performance is too far away from what is required for modern 3D game engines (this is commonly accepted theory, rather than a fact, but The Computer Language Benchmark Game (Fulgham & Gouy, 2014) which tries to benchmark language performance by measuring efficiency of implementations of selected programs, is an indication of its truth), due to garbage collector and lazy evaluation overhead and tendency to run into time and space leaks.

John Carmack, a games industry veteran, pointed out the benefits of following some FP concepts in C++ in games development, without necessarily jumping into full-blown FP language such as Haskell or Lisp (Carmack, 2013; Carmack, 2012). Tim Sweeney drew similar conclusions in his presentation focused on languages used in games development (Sweeney, 2006). Furthermore, another industry veteran Mike Acton in his presentation (Acton, 2014) makes a very good case for Data-Oriented Design in C++, which is exactly how experienced functional programmers think about code – put data first and use functions as transformations on them (Callaghan, 2012). Further research of Data-Oriented Design is therefore one of the secondary project's objectives.

Therefore the focus of this project was to find out if it would be possible to use a selection of FP concepts in programming language C++, which allows for very efficient implementation, to a perceivable benefit without sacrificing too much performance, and if so to find out which concepts are these.

1.2 Project Aims and Methodology

This project sets out numerous secondary and primary objectives, which are listed in this section, following an explanation of the aim of the project, including the research question, and summary of its outline.

1.2.1 Aim of Project

As the complexity of source codes of mainstream 3D game engines is growing every year, many developers find that the tools they are currently using are getting less and less efficient at solving their problems. The aim of this project is to develop a limited 3D game engine in order to investigate if any of the FP concepts can aid in improving the

game engine development in C++ by making the source code more concise, modular, testable and reliable, with simplified concurrency and rapid prototyping.

1.2.1.1 Research question

Can functional programming concepts be used in traditionally imperative language C++ to better manage complexity and thus improve 3D computer game engines development?

1.2.2 Brief Outline of Project

This project had involved initial research into functional programming concepts in Haskell. The FP concepts learnt from Haskell were individually evaluated for their potential suitability, by assessing their advantages and disadvantages as well as considering how they can be implemented in C++.

Two implementations of a limited (by time and resources) 3D game engine were developed in C++, one traditionally imperative and one functional. The selected subset of FP concepts was applied to C++ code in the functional implementation.

To compare the implementations against each other, numerous factors were considered – number of lines of code, code modularity and reusability, maintainability, testability and reliability, approaches to implementing concurrency, and building a prototype using the engine. Most of these qualities cannot be simply measured and therefore the conclusion of their suitability will be more open to discussion.

1.2.3 Secondary Project Objectives

These were the secondary project objectives:

- **Assess the FP concepts for their suitability for games development** – Some concepts might be more suitable than others. By distinguishing concepts that do not pass requirements for computer games development, more resources can be spent on implementing the ones that are potentially beneficial. One of the main criteria used in this objective was performance cost. As mentioned in section 1.1 performance is very important for game engines.
- **Research how to implement selected FP concepts in C++** – This project can build upon previous attempts at using FP concepts in C++, which need to be researched before the development can commence.
- **Gather and study resources on 3D game engine development** – The outcome of the project will depend on how well designed the 3D game engine is. Designing game engine is not a trivial task, therefore it will be crucial to learn as much as possible from research that has already been carried out by others.
- **Research Data-Oriented Design** – As state before, Data-Oriented Design is recommended style for developing games and game engines by one of the industry leading developers. Its procedures are very similar to the way applications are being developed in Haskell.

1.2.4 Primary Project Objectives

The primary project objectives were as follows:

- **Design and develop a limited 3D game engine using selected FP concepts** – This version of 3D game engine was developed in C++ that strictly followed the selected FP concepts.
- **Develop a limited 3D game engine, with equal capabilities as the FP version, using traditional imperative style** – This implementation was used as a reference to be compared against the functional implementation.
- **Build and run tests for comparison of the two implementations where applicable** – The tests mainly comprised of profiling to measure performance trade-off of FP concepts, if any.
- **Produce game engine demonstrations** – These demonstration applications were developed in a way that promoted discussion of the research question of this project on qualities that cannot be directly measured. For example, one use can be implementing a change in the engine to find out how maintainable it is, or implementing a prototype to see if FP brings any benefits on this front over the imperative implementation.
- **Analyse the data produced by the tests and discuss the results** – The quantitative data are presented using tables and graphs for an easy comparison. The conclusion included a critical discussion of each of the selected FP concepts and their benefits and setbacks.

1.2.5 Hypotheses

Based on the results reported from projects that adopted the Functional Programming paradigm, the following three hypotheses have been developed:

H1: Functional programming concepts of first-class and higher-order functions, currying, immutable data, purity, algebraic and recursive data types, recursion, lazy evaluation, lambda expressions and closures can be used in C++ for game engine development.

H2: Functional programming concepts can be used in C++ game engine development to increase productivity, promote code modularity, extensibility, testability, reliability, simplify concurrency, and rapid prototyping.

H3: Functional programming concepts used in C++ game engine development will have negative effect on code performance.

The accuracy of these hypotheses is resolved by the end of conclusion of the project.

1.3 Report Structure

The following chapter 2 discusses the researched problem and the proposed solution backed up by literature review. The plan of the project methodology, which was test and develop coupled with iterative software development lifecycle, and a gathered list of requirements are detailed in chapter 3. Chapter 4 discusses in detail design and implementation of both versions of the game engine, the chapter's structure follows the iterations of the project's lifecycle, with substantionaly more focus on the functional programming version, which represents the proposed alternative solution to the researched problem. Chapter 5 discusses the testing and evaluation techniques employed in this project, mainly driven by development of demonstration applications implemented in both versions of the game engine. Finally, chapter 6 then finishes off

with summary of project's achievements and conclusions, but also its limitations, which offer possible research areas for further research.

Because one of the main focuses of this project is C++ language, this report contains numerous figures of code snippets, whose format is shown in Listing 1.1 below. These code snippets help to explain and illustrate certain topics alongside mathematical expressions that are written in format $A = B + C$.

```
void( *fnPtr1 )( int );
```

Listing 1.1 Example of code snippet format

The source code for the project is publicly available as are the data measured from the profiling of final applications. The resources obtained for the project are credited in Appendix A, the specifications of the profiling machine and the compiler settings can be found in Appendix B with the profiling data illustrated in Appendix B. The project repository is detailed in Appendix D together with explanation of the directory hierarchy. Appendix E lists the full source code of the functional programming version, and the following Appendix F contains source code of the imperative programming version.

2 Literature Review

This chapter is split into three sections each dedicated to one of the secondary project objectives identified earlier:

- Assess the FP concepts for their suitability for games development
- Research how to implement selected FP concepts in C++
- Gather and study resources on 3D game engine development
- Research Data-Oriented Design

2.1 Assess the FP concepts for their suitability for games development

In this section the selected FP concepts are assessed for their suitability for games development based on a single most important criteria, their performance cost, weighed against their practicality.

2.1.1 Evaluation

2.1.1.1 First-class & Higher-order Functions

Functions in C++ are first-class citizens and can be operated on by obtaining a pointer to a function, which can subsequently be used to call the function or pass it as an argument to another function. In C++ pointers are essentially memory addresses of the machine on which the code runs (Stroustrup, 2013), meaning there is no abstraction overhead when using pointers and their practicality is high. C++ prohibits modification of functions by overwriting the memory address of the function which makes them safer to use.

2.1.1.2 Currying

Overhead of currying depends on the arguments we pass to the partial application of a function, because the function has to create a copy of each argument. It is however possible to reduce the overhead by passing a reference instead of a value or remove it altogether by using the `move` operation from the Standard Template Library (abbr. STL) on the function arguments (Wakely, 2013; Gockel, n.d.). Currying can help promote reusability of code and thus improve its concision.

2.1.1.3 Immutable Data

Immutability has many implications, both positive and negative, however the benefits arguably outweigh the drawbacks. The memory footprint will obviously have to increase and the data structures (lists, trees, queues, etc.) have to be re-implemented as well as the operations on these data structures (common iteration constructs such as `for` and `while` loops can no longer be used, the iteration in FP is usually expressed as recursion) to be immutable too as covered in the text Purely Functional Data Structures (Okasaki, 1998). The immutable data structures can cleverly reuse previous versions to reduce the increase in memory footprint, but they cannot remove it completely. For that price one can gain thread-safe code by default, thus it becomes easier to write, use and reason about the code and on top of that the data structures become persistent and support multiple versions, rather than being ephemeral, which is a data structure that allows only a single version at a time (Okasaki, 1998).

On the other hand, there are data in a game engine such as framebuffer where the cost of immutability would result in very poor efficiency, but as long as the operations on the "unsafe" data are separated from the "safe" parts of the code much in the same way that Haskell demonstrates (Lipovača, 2011), one can still benefit from immutability.

It should be emphasized that the expected simplification of concurrency adoption is deemed to be very important, as it can quite possibly outweigh performance loss implied by immutable data (Tulip, et al., 2006).

2.1.1.4 Purity

Purity of functions is loosely linked to immutable data and is a very important for FP as it enforces referential transparency. For functions in which using immutable data would be too inefficient, one can relax the purity rule to allow mutation of local data inside the function body, as long as the data or reference to the data that's being mutated does not escape the function's body and does not cause any side effects and still profit from purity in exactly the same way that Haskell does. The benefits are similar to those of immutable data, thread-safety and improved ability to reason about code. It was named as an essential property of Haskell that even lead to a parallel implementation of GHC (Glasgow Haskell Compiler) runtime system for running on a shared memory multiprocessors in around year 2009 (Marlow, 2013).

2.1.1.5 Algebraic & Recursive Data Types

The Algebraic data type system is a powerful tool and it will certainly be worth the small additional performance overhead it might cost. Recursive data types are a convenient way to define dynamic data structures and in FP they are usually used simultaneously with immutability, even though they can be mutable as well. It would be incorrect to try to compare FP recursive data types to their imperative counterparts when taken out of context, both of them will have a place where one performs better than the other and to be able to make a sound judgement, this comparison needs to include their use case scenario. In most cases the imperative analogue would be preferable over the FP ones if only their isolated implementation running on a single thread of execution is considered, but when the complexity of the whole application running on multiple cores is included in the reasoning, it might suddenly turn the tables the other way.

As an example let us consider the doubly-linked list implementation as suggested by Patrick Wyatt, a long-time game developer, presented in his series of blog posts (Wyatt, 2012). The given implementation of linked lists called intrusive lists, which is widely used in C, is clearly optimal, both in performance and reliability and it should be preferred over the C++ standard library list and Boost library intrusive list. However, this implementation comes with a caveat which gets mentioned in the post as well – multi-threaded code. Patrick demonstrates couple of solutions for writing thread safe deletion of an element from the linked list, which incorporates the use of a mutable exclusion object (commonly referred to as mutex) and comes with many caveats itself (Ljumovic, 2014; Williams, 2012). All these issues can be avoided with immutable shared data (Sutter, 2014).

2.1.1.6 Recursive Functions

As mentioned earlier in the Introduction chapter, recursive functions that perform tail-call recursion can be optimized in C++ resulting in the same or almost the same performance one would get from using imperative iteration constructs. Even though tail

call optimization is not a part of C++ standard, the most common C++ compilers (VC++, GCC, LLVM's Clang, Intel C++ compiler) support tail call optimizations when the optimization flags are switched on and only under certain circumstances (Balaam, 2012; Smith, 2006; Bauer & Pizka, 2004; Probst, 2001).

A typical game engine consists of one main loop with series of operations that make up the whole game (McShaffry & Graham, 2012). Using recursion in C++ for this loop is not possible and would lead to stack overflow, because the loop will not terminate until before the game exits and as such cannot be optimised. Tail call optimization can mostly only be applied for relatively small and simple functions.

2.1.1.7 Lazy Evaluation

Lazy evaluation brings two potential benefits, infinite data structures and improved efficiency of certain algorithms by only doing a work that is absolutely necessary (O'Sullivan, et al., 2008). In Haskell this is achieved by building up *thunks*, which are deferred computations which can be evaluated at any point (Lipovača, 2011). This creates overhead and makes reasoning about codes performance and especially about the space (memory) requirements very difficult which makes it a common source of controversy in Haskell (Peyton Jones, 2003), because the programmer has to sacrifice any direct control over the order of program's execution (Hughes, 1990). The ability to reason about performance of code is too important in game engine development to be able to trade it away for lazy evaluation therefore this concept will not be used for the purpose of this project.

2.1.1.8 Lambda Expressions & Closures

Lambda expressions are compiled like all other C++ code without any overhead cost left to the runtime meaning they perform just like any other function. Closures are well optimized and as such can be used to make the code more concise. The performance of a lambda expression being passed as an argument to a function is typically identical to an equivalent code where the expression is inlined inside the function's body (Stroustrup, 2013).

2.1.2 Summary

From the evaluation results it has been concluded that first-class and higher-order functions, pure functions, lambda expressions and closures will be fully adopted to make the best of FP concepts. On top of that immutable data, recursive data types, recursion and currying will be utilized in places where justified, leaving lazy evaluation to be the only FP concept that will definitely not be adopted. This will involve further assessment of suitability of a solution to a concrete problem.

2.2 Research how to implement selected FP concepts in C++

In this section, possible implementation methods for the FP concepts that passed the previous stage of evaluation are explored. For the concepts that have multiple alternative implementations, further comparison of their suitability is required.

There are multiple initiatives that try to utilize FP concepts into C++ that will be considered in this section. First of all; new C++ standards bring FP concepts natively into the language, most of which can be found in the Standard Template Library *functional* header (Horton, 2014; Deitel, et al., 2013; Josuttis, 2012). There are also numerous libraries that implement their selected FP concepts such as Boost functional (Polukhin, 2013), functional-cpp (Haffmans, 2013), Functional Programming in C++

built using Boost libraries (Sankel, 2010), and most likely the earliest attempt to bring FP concepts to C++ called FC++ (McNamara & Smaragdakis, 2000). Interestingly, many concepts in the new C++ standards are, in fact derived from Boost library collection (Deitel, et al., 2013).

2.2.1 Research

2.2.1.1 First-class & Higher-order Functions

C++ supports first-class and higher-order functions "out of the box" in form of raw pointer to function (Stroustrup, 2013). An example of a raw pointer to a function whose argument is `int` and return type `void` can be seen in Listing 2.1 below. This approach has superior performance when compared to the alternatives – wrapper type `std::function` from the Standard Template Library (Listing 2.2) and from Boost library, which are slightly more complex and stateful. However, they are more powerful as they can hold any sort of callable entity such as callable object (Polukhin, 2013; Josuttis, 2012). One such type of callable object in C++ is the Lambda Expression. The actual performance cost was found to be 3 CPU instructions for the raw function pointer and 5 CPU instructions for the `std::function`, which can be verified by compiling the source code from the discussion about this topic (Trapni, 2013).

```
void( *fnPtr1 )( int );
```

Listing 2.1 Raw function pointer in C++

```
std::function<void( int )> fnPtr2;
```

Listing 2.2 Function wrapper from STL

2.2.1.2 Currying

C++11 standard has added support for partial application by using the `bind` function from the Standard Template Library, which has been derived from the Boost's `bind` function (Polukhin, 2013; Josuttis, 2012). An example of usage of this function can be found in Listing 2.3 below. In this snippet, number 3 is partially applied to the last argument of the function `f`, which creates the new function `f1`, that requires two parameters. Its first argument is bound to be second argument of the newly created function `f1` and its second argument is bound to the first one by use of placeholders.

```
1.  void f( int n1, int n2, int n3 )
2.  {
3.      std::cout << n1 << ' ' << n2 << ' ' << n3 << std::endl;
4.  }
5.  int main( )
6.  {
7.      auto f1 = std::bind( f, std::placeholders::_2,
8.          std::placeholders::_1, 3 );
9.      f1(1, 2);
10. }
```

Listing 2.3 Partial application in C++

2.2.1.3 Immutable Data

In C++ immutable data can be declared with `const` keyword, which prevents any modifications of the data after it has been initialized (Stroustrup, 2013), unless one was to use `const_cast` operation, which will be prohibited in this project.

2.2.1.4 Purity

Naturally it is possible to define pure function in C++ as it is just a regular function without any side effects. The tricky part is that there is no simple way of enforcing a function to be pure just like Haskell functions or functions in D programming language annotated with `pure` keyword (Alexandrescu, 2010). It then comes down to the programmer's discipline or use of some external tool such as static code analyser or compiler function. For example GCC supports this through `__attribute__ ((pure))` function annotation. Property which is in C++ commonly referred to as "const-correctness" might at first seem to have the same effect, however under closer inspection it becomes obvious that it is not sufficient, as it still allows access to a mutable global state (Reddy, 2011).

Generally, not all functions can be pure, so to get the benefit that pure functions offer, it is crucial to separate the pure parts from the impure part (i.e. functions with side effects). In Haskell impure functions have their return type wrapped in `IO`, `SafeIO` or `ST` type. In C++ this can be done with help of prefixing or appending impure function names with a certain string, which can be error-prone. An alternative solution would be to wrap return type in `IO` type, similarly to Haskell, following the example of one of the previously listed C++ FP libraries (Sankel, 2010).

As a side note, in programming language D (which was hugely influenced by C++), pure functions are allowed to break the purity rules with operations marked with the keyword `debug`, which is a very convenient addition (Alexandrescu, 2010).

2.2.1.5 Algebraic & Recursive Data Types

Modern Functional Programming in C++ paper demonstrates an elegant solution on how to implement basic algebraic data type system (Sankel, 2010) with help from Boost libraries. There is a planned feature for C++ called "Concepts" (Gregor, et al., 2006), that would be beneficial in algebraic data types definitions as they are very similar to type classes in Haskell (Bernardy, et al., 2008). The "Concepts" were proposed for C++11 standard, however because they were considered broken (Kalev, 2009) the feature has been removed from the draft and so far it is only available as GCC extension (Gregor, 2011).

Recursive data types in C++ can be implemented simply by use of pointers. Perhaps the best example of recursive data structure is a list, which is used heavily across the many of the FP languages. It should then come as no surprise that a popular FP language Lisp derives its name from "list processing" (Seibel, 2005).

The recursive list commonly comes with these core functions for list construction and for accessing its elements. It is astounding that just by composition of these four operations it is possible to build almost any more complicated functionality for lists (O'Sullivan, et al., 2008):

- `[]` – an empty list

- `(:)` – operator which adds an element to the front of a list (usually called `cons`, short for construct)
- `head` – function that returns first element of the list
- `tail` – function that returns the list minus the first element

To be able to operate on the recursive data types more comfortably Haskell uses technique called pattern matching. The possibility of using pattern matching in C++ has been well researched, resulting in a release of open pattern matching library for C++ titled Mach7 (Solodkyy, et al., 2014).

2.2.1.6 Recursive Functions

C++ language has native support for recursive function calls. As discussed in the previous section, recursive functions have to be tail-recursive to avoid an extra performance penalty and possibly causing stack overflow. However, there is not any simple method to perform check that a function is tail-recursive and that the optimization will be applied, other than a manual check and therefore the correct use of tail-recursion in C++ relies on programmer's discipline.

```
1. unsigned int factorial( const unsigned int x )
2. {
3.     if ( x == 0 )
4.     {
5.         return 0;
6.     }
7.     return factorial( x - 1 );
8. }
```

Listing 2.4 Tail-recursive function

2.2.1.7 Lambda expressions & Closures

The C++11 standard brought native support for lambda expressions into C++ (Deitel, et al., 2013). But even before C++11 it was possible to utilize lambda expressions using Boost library (Polukhin, 2013).

```
1. auto add = []( const int a, const int b )
2. {
3.     return a + b;
4. };
```

Listing 2.5 Lambda function for adding two numbers

C++ supports closures in form of lambda expressions that can capture values of references to values from outside of its scope using a capture list (Stroustrup, 2013).

```
1. int five = 5;
2. auto addToFive = [five]( const int x )
3. {
4.     return x + five;
5. }
```

Listing 2.6 Closure that captures a value of variable outside its scope

2.2.2 Summary

It has been established that most of the FP concepts can be natively implemented in C++, especially when targeting C++11 standard or higher, and in some cases with help of Standard Template Library. There is no need to resort to Boost library, which is considered to be too heavy for performance critical code anyway and is generally not being used in 3D game engine development (Fleury, 2014).

2.3 Gather and study resources on 3D game engine development

Game developers research papers are rare and game conferences do not publish journals either, so the most of the up-to-date information about the direction of game development can only be found through alternative media such as blogs, conference recordings and even social media.

Typical game engines are very complex application with large code bases, a proof of this is Unreal Engine version 3 with over 2 million lines of code (Lowensohn, 2010). This version was in fact released first time in year 2004 and it is now exceeded by the version Unreal Engine 4.

Five main resources helped to gain better understanding of traditional imperative game engine development techniques. First four texts on 3D game engine development (McShaffry & Graham, 2012; Harbour, 2010; Eberly, 2007; Sherrod, 2006), while the last one covers graphics programming with DirectX 11 (Sherrod & Jones, 2011). For the functional programming version the main resource was documentation of Yampa embedded domain-specific language (abbr. EDSL) and description of demonstration on FRP approach used in process of building of this clone of the classic game Space Invader in Haskell (Courtney, et al., 2003).

2.3.1 Game Engine Architecture

A typical game engine is built from separate building blocks that encapsulate their dedicated functionality. This list is not exhaustive, because as stated above, game engines are very complex systems with multitude of functionalities built by large teams of seasoned professionals. There are many parts that would be very time consuming to develop and so these are left out the scope of this project.

2.3.1.1 Application Layer

The application layer is usually composed of the most of the input/output functions such as initialization and shutdown, the main loop, handling devices of input, file systems, resources, threads, allocation of memory, measuring time, handling window, data structures implementations and more (McShaffry & Graham, 2012). Application layer developed for Window platform will make a heavy use of Windows API.

2.3.1.2 Actor System

There are multiple ways to structure actor or game object architecture.

Component Based System

Component based game object architecture is a considered to be an optimal design as it decouples actor specializations in a manageable and extendable way where actor's components are allowed to change on the runtime and that makes it easier for game designers to tweak their game object's functionality (McShaffry & Graham, 2012). But there are disadvantages with using component based system as well. It is not well suited for functional programming as it is designed around mutation of state of the actor and its components.

Functional Reactive Programming

In imperative C++ users of a game engine typically implement their desired functionality by extending polymorphic classes. In FP this is done via higher order functions instead, by having the API accept user defined functions which contain the desired functionality. This is a better approach because polymorphic classes harm reusability and don't map to real problems well as has been argued in many sources (Albrecht, 2009), however it often suffers from its own issues such as time and memory leaks, even though these issues have been actively researched and their possible solutions found (Krishnaswami, et al., 2012).

An alternative to component based system that has been popularised in functional programming and also used in most of the games and interactive applications written in functional programming style is Functional Reactive Programming (abbr. FRP). There are only a few games written in Haskell however for the most of them it is true that they use some form of FRP, which is most often implemented in Haskell embedded domain-specific language (abbr. EDSL) called Yampa. For example Frag is a clone of a classic game Quake 3 and it is arguably the most advanced computer game implementation in Haskell up to date, Yampa Arcade is another clone of the classic Space Invaders game and Cuboid that has been written in just about 250 lines of code is a small puzzle game.

Instead of using the conventional component based actor architecture, Functional Reactive Programming based architecture is a better fit for functional programming style. FRP and its design and implementation in this project have been discussed further in sections 4.1.2.10 and 4.2.2.10.

2.3.1.3 Maths and Geometry

This part involves calculations that most of the time involves vectors, matrices and quaternions operations such as vector distance, length, dot product, cross product, normal, radian conversion, degree conversion and linear velocity (Harbour, 2010).

2.3.1.4 Model and Material

Game engine has to be capable of importing and using 3D models and materials through the renderer, to provide a way to build custom 3D scenes to the users of the engine.

2.3.1.5 Renderer

The renderer draws 3D scene onto a render target. In a modern game engine, the renderer is predominantly shader based, which gives total control over the graphics

pipeline to developers and consequently allows maximizing the potential of powerful GPUs (Sherrod & Jones, 2011). Rendering process is typically using double buffered swap chain. Double buffer consists of front and back buffer. The renderer renders into the back buffer, after which it is swapped with front buffer to be presented on the screen.

The two low-level API options for renderer implementation are DirectX and OpenGL and each is supported on different platforms. Both versions also use different shader language.

2.3.1.6 Camera

A typical game engine has some notion of a camera, a concept that is being used to control what is visible on the screen. A game engine's camera is either orthographic or perspective, however both them can be defined through one common implementation. This implementation defines camera's frustum which consists of six "clipping" planes that can be used to find out whether an object is inside camera's view. If the near and far clipping planes are the same size, it means that the camera is orthographic. A perspective camera's near clipping plane is smaller than its far clipping plane.

2.3.1.7 Scene Graph

Scene graphs are the means of organizing the scene objects into a hierarchy, that significantly help improve game engine performance by culling non-visible objects out of the rendering pipeline. There are multiple ways of organizing scene graph and there is no one perfect solution as the efficiency largely depends on the type of the game. Two basic ways of arranging scene graph are based on spatial proximity and render-state coherency (McShaffry & Graham, 2012; Eberly, 2007).

2.3.1.8 Collision Detection & Physics

Physics simulation in a game engine is one of the most complex parts and as such it is usually self-contained in its own physics engine and it is a common practice to adopt one of the available physics engines such as PhysX and Havok (McShaffry & Graham, 2012).

2.3.2 Summary

The gathered game engine development resources proved to be sufficient to guide the author through the development process and serve as reference when required.

2.4 Research Data-Oriented Design

Data-Oriented Design as shown by Actor (2014), is a design and development process, that recommends to primary drive design by data, and logic secondary. This apparently clashes with design methodology that is ingrained from OOP, where the software architecture is designed around classes and classes are designed around their functionality, i.e. logic. The way Data-Oriented Design organises its memory layout around data leads to very efficient code performance.

The researched texts on Haskell indicated (Mena, 2014; Marlow, 2013; Callaghan, 2012; Lipovača, 2011; O'Sullivan, et al., 2008; Hutton, 2007) that the way applications are developed is by defining data first and then logic as pure (free of side-effecs) functions that transform the data.

2.5 Conclusion

Because the reuse mechanism in imperative and especially object-oriented systems is structural and involves interactions that change the internal state, it typically creates coupling that harms the ability to reuse. That created a whole movement of design patterns, which only tries to make up for shortcomings of involved languages. On the other hand FP promotes reuse on coarser-grained function level through composition, which is based on mathematical category theory, and it reported to be much more effective (Ford, 2014).

The assessment of suitability and research of implementation techniques of selected FP concepts together with all the studied game engine development resources provides a solid ground on which to build the desired software, which is subsequently expected to prove to be a reliable evaluation framework of suitability of FP concepts in C++ game engine development.

3 Problem and Systems Analysis

This chapter discusses chosen research methods that were concluded based on research detailed in the previous chapter. It then sets out both functional and non-functional requirements that were important for finding answers to the project's research question and hypotheses as has been indicated by the reseach of the problem area discussed in section 2.2.1.

3.1 Overall methodology

This project aimed to investigate whether the issues developers currently have to face when developing a 3D game engine using traditional iterative approach could be elevated through adoption of functional programming principles in C++. It has been demonstrated earlier that functional programming is already successfully integrated in many areas of software development. Even then the computer games development and especially game engine development is still dominated by the imperative paradigm and usage of FP concepts has mostly been evaluated only theoretically. Therefore this project will use **develop and test** methodology to investigate suitability of using FP concepts in C++ game engine development to step away from the theory and to focus on actual game engine development perks and issues this approach might bring.

Development of two versions of engine, functional and imperative, will provide evaluation means through which the project's research question can be discussed and answered and the hypotheses can be tested for their validity.

3.2 Development lifecycle

Because the architectural design was mainly driven by the data and was likely to evolve during the development, the iterative design method has been selected to accommodate reiterations over the implemented functionality (Tsai, et al., 1997).

Figure 3.1 illustrates the stages of iterative design and testing lifecycle. The whole process starts with *Requirement Analysis* in which both functional and non-functional requirements are researched. These are then formulated in *Requirements Specification* stage, after which the desired functionality is known and therefore *Systems Tests* can be designed. The project structure is then decided in *Architectural Design* stage, which is concerned about the individual modules and their interactions. When this stage is over, *Integration Tests* can be designed to confirm how the modules co-operate with each other. In the next stage, a *Module Specification* is decided together with its *Unit Tests* that check its functionality. After that the actual *Coding* stage begins. It is then followed by execution of all the test stages from bottom up. A failure at any test stage leads to reiteration of the related design or specification stage. Once all the tests are passed the development lifecycle is over and the piece of developed software thoroughly tested to be conforming to all its requirements.

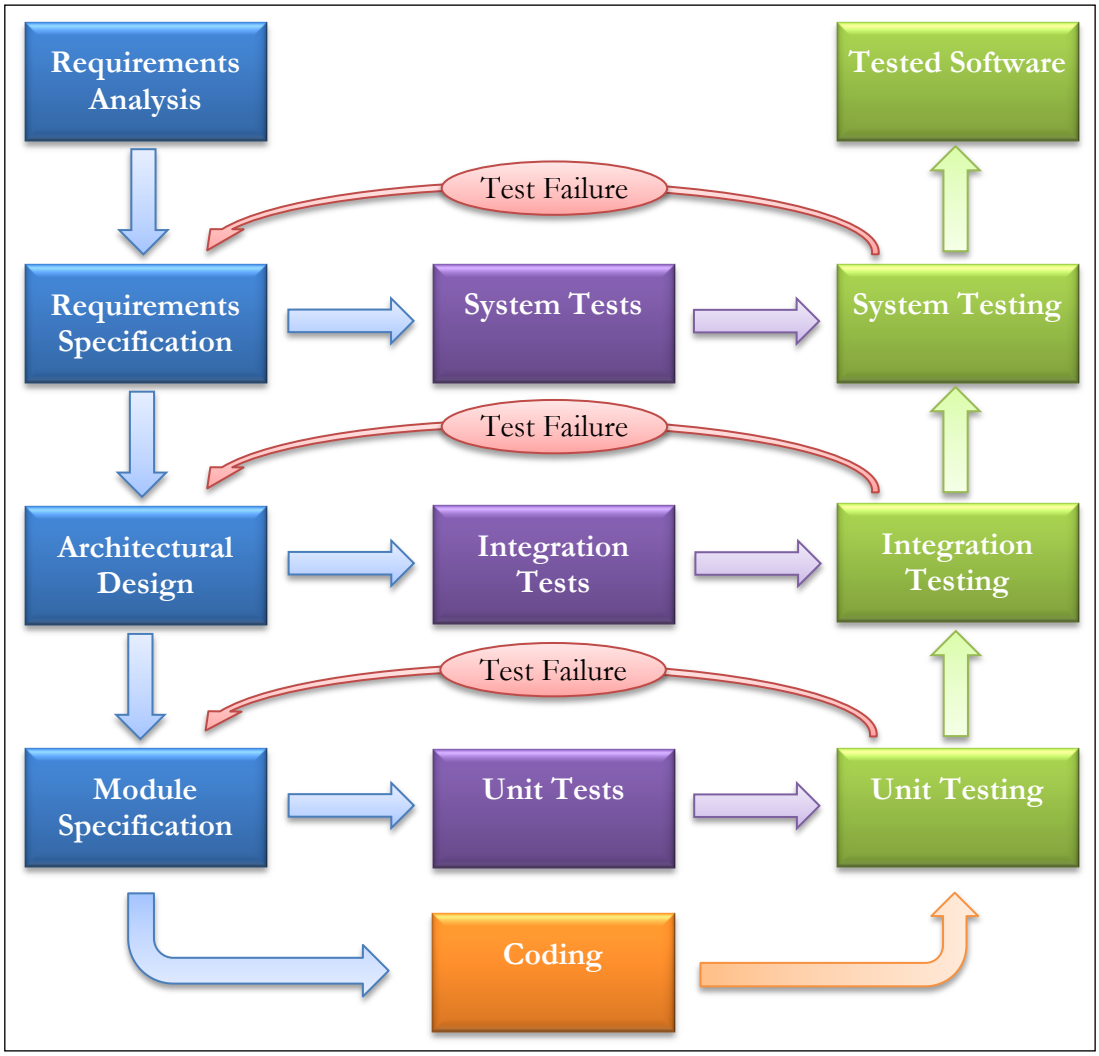


Figure 3.1 Iterative design and testing lifecycle

3.3 Requirements Analysis

Requirement Analysis stage is essentially derived from the section 2.3 in Literature Review chapter, where the main game engine components were researched in detail.

Naturally, it would not suffice to implement a single version of the game engine using FP concepts, simply because there would not be anything to compare it to and thus assess how well the FP concepts map to C++. Comparison to any other existing open source engine would be impossible unless exactly the same functionality was to be implemented and even then there are other aspects, such as architectural design and code quality, which would most likely influence the results in favour of one solution or the other.

To gain more accurate results that would compare FP code against typical imperative code with minimal impact of the other aspects, it has been decided that two versions of a 3D game engine with equal capabilities are to be designed and produced, the first one will strictly follow selected FP concepts from secondary research, the other one will be written in traditional imperative style. The first two objectives will be worked on simultaneously, to ensure that they are always in a similar stage of completion.

3.3.1 C++ Language Analysis

3.3.1.1 C++11 Standard

VC++11 compiler used supports the following C++11 Standard features (Microsoft, 2015) that extended the core language and the STL, which were used in this project to improve the code quality (Deitel, et al., 2013):

- Lambda Expressions
- Rvalue References and Move Semantics
- Initializer Lists and Uniform Initialization
- Type Inference
- Range-based For Loop
- Trailing Return Type
- Null Pointer Constant – `nullptr`
- Strongly Type Enumerations
- Template and Type Aliasing
- Variadic Templates
- Explicitly defaulted and deleted special member functions (however, VC++ does not support defaulted or deleted move constructors and move assignments operator).
- Static Assertions

3.3.1.2 C++14 Standard

None of the C++14 Standard new features that are supported by VC++ compiler were beneficial for this project and therefore were not used.

3.3.1.3 Template metaprogramming

The template metaprogramming in C++ is notorious for its difficulty to debug caused by cryptic compilation error messages, because in C++ the technique of using templates to perform compile-time execution and Turing completeness of templates that has led to its rise have been discovered by accident (Veldhuizen, 1995; Stroustrup, 1993).

It could be more comfortable with use of proposed feature of C++ called "Concepts", that allows to constrain template type parameters without any runtime overhead (Stroustrup, et al., 2013). The idea of "Concepts" is very similar to Haskell's classes and would greatly simplify any effort to implement Haskell concepts in C++. Unfortunately, this feature has been postponed in the standard and so far it has only been implemented as an extension in GCC compiler. VC++ lacks support for it and therefore it could not be used in this project. The lack of "Concepts" support can to a limited extend be substituted with many other C++ constructs that support template metaprogramming to improve its ease of use and safety such as `static_assert`, `decltype`, `constexpr` (unsupported in VC++ therefore unused), plus those defined in STL *type_traits* header file.

3.4 Requirements Specification

As mentioned earlier, the *Requirements Specification* is derived from the *Requirement Analysis* stage. The requirements are divided into either functional or non-functional

requirements (not to be confused with functional programming and non-functional (in here imperative) programming which are entirely different concepts). The set of functional requirements dictates the required behaviour of an application. On the other hand, non-functional requirements relate to properties that do not directly affect functionality of an application (except for extreme cases), but can have an effect on its quality.

3.4.1 Functional Requirements

It has been established that the functional and imperative implementations will have exactly the same functionality and therefore the following functional requirements apply to both of these versions. Based on the Literature Review section 2.3, the game engine should be able to:

- Provide data structures and associated operations for mathematical and geometrical constructs such as vector, quaternion, matrix, plane, camera's frustum and colour
- Open and close a configurable window
- Handle input from keyboard and mouse
- Render 3D models using HLSL shaders
- Keep a track of time
- Provide an interface for populating scene with actors
- Provide an interface for customising 3D model, material, transformation and behaviour of actors
- Provide pre-built primitive 3D geometry such as cube or sphere
- Load custom 3D models and materials
- Manipulate the scene's camera

3.4.2 Non-functional Requirements

Based on the research done into the problem, the non-functional requirements for both the functional and imperative implementations are:

- Resources such as 3D models and materials should be reusable once loaded to avoid unnecessary memory wasting
- The implementations will make use of features from C++11 Standard whenever possible

The non-functional requirements for the FP implementation are:

- It should be implemented through the FP concepts selected in Literature Review section 2.1.1 and 2.2 to serve as an evaluation tool for these concepts
- It will not use OOP constructs, instead a Data-Oriented design will be adopted that in C++ implies C style code, this is justified by the research detailed in section 2.4
- The logic will be implemented through higher-order functions

And finally non-functional requirements for the imperative implementation are:

- The imperative implementation will be written in OOP style to contrast the FP version
- The logic will be implemented through mutable state and class hierarchy as this is the traditional imperative style
- It will provide classes that the users of the engine can inherit from to implement their desired functionality

3.4.2.1 Technical Specifications

The targeted platform is Windows 32 and 64-bit with graphics support for DirectX 11. For development Visual Studio 2013 IDE has been used with primary compiler being Microsoft® C/C++ Optimizing Compiler Version 18.00.31101. The C++ exceptions and RTTI (Run-Time Type Information) were disabled, as this is the common practice for performance critical code (Fleury, 2014). The impact of having RTTI disabled is that `dynamic_cast` cannot be used, which is only useful for polymorphic classes and even then not necessary and therefore not important for this project.

Directory Structure

Both functional and imperative versions of the project are organised into the directory structure shown in Appendix D.

4 Design and Implementation

It has been discussed earlier that FP applications are designed and developed in Data-Oriented style, which will also be the methodology used in this project. Data-Oriented design leads to a highly efficient code, because good memory layout reduces the chance of cache misses and the need for the multiple levels of indirection that can for example be seen in Object-Oriented design (Acton, 2014). In Data-Oriented design the primary focus is on data, secondary focus is on transformations (i.e. functions) that transform input data into output data. In other words, the flow of the program is driven by the data.

This chapter describes the *Architectural Design*, *Module Specification* and *Coding* stages of the iterative software development design and testing lifecycle that has been set out in section 3.2. It is organised into sections that correspond to the iterations involved in development of the project.

As the engine has been built from ground up, the goal of every iteration was to develop a selected modular part of the engine (such as timing facility, graphics rendering, input processing, etc.), beginning with the window and renderer that allow to test functionality of the other modules by program execution, then progressing towards the other modules. In the beginning of many iteration stages the overall engine architectural design has been revisited. To illustrate the point, the Figure 4.1 shows the flow of individual iterations.

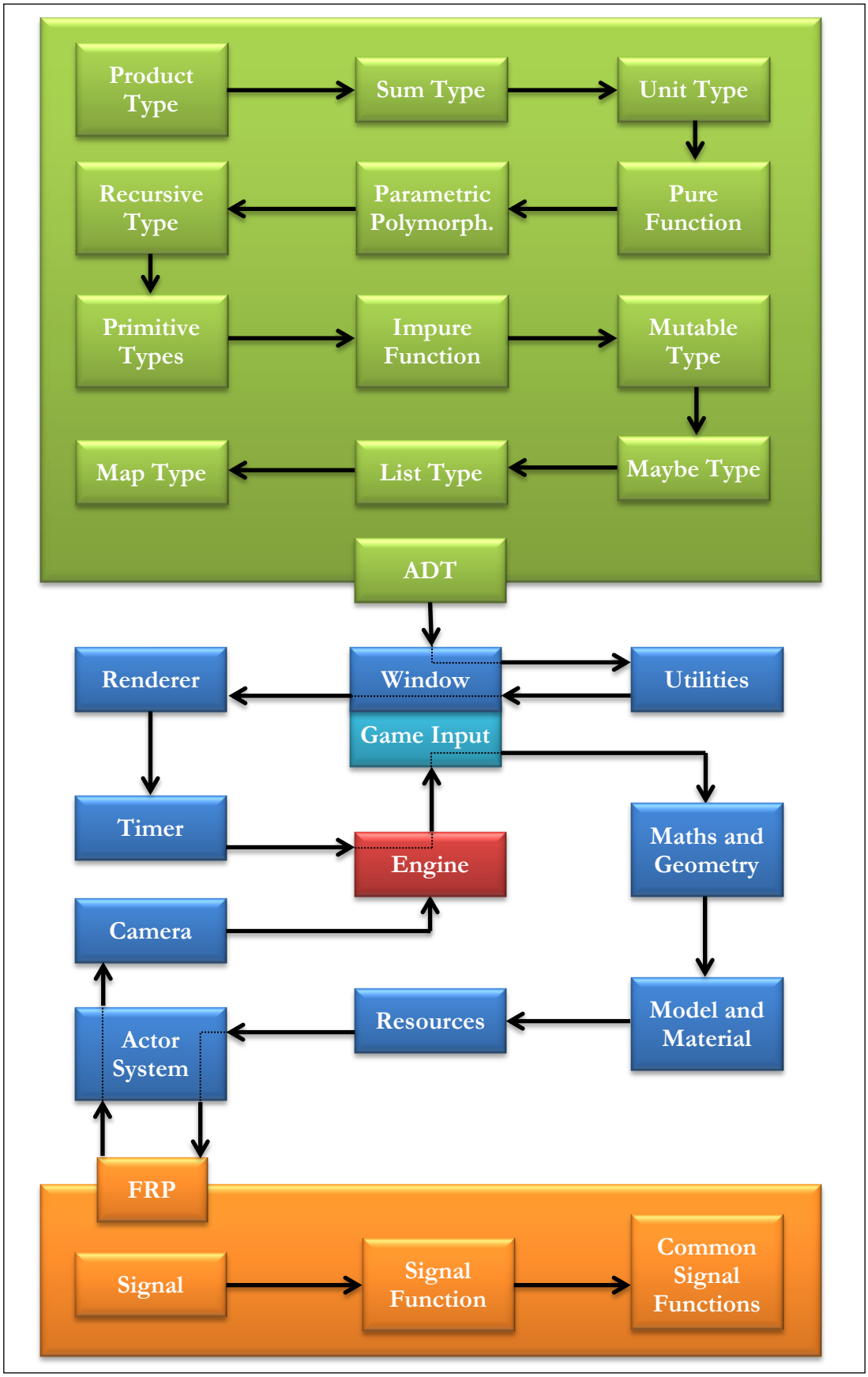


Figure 4.1 Iterations of design and implementation

4.1 Design of Functional Programming Version

To follow the Data-Oriented design that has been justified in the beginning of this chapter, the following subsections will be driven by definition of data. The design is formally defined in language that is independent of implementation programming language. The following sections follow the flow of the diagram from Figure 4.1.

4.1.1 ADT - Algebraic Data Types

In the first iteration, a simple algebraic data type system has been designed and developed, as the parts of it were used as building blocks for the rest of the game engine. The Algebraic data types in Haskell are based on type and category theory and therefore can be formally described in terms of mathematics. Most of the following sections are interlinked and build up on knowledge from their preceding sections.

4.1.1.1 Product Type

The type $A \times B$ is a Product of two types A and B that contains elements of both of these types, in other words the pair (a, b) is of type $A \times B$ if a is of type A and b is of type B. Product type, usually called tuple, can contain more than two elements, in which case we are speaking about *n*-tuple, where *n* is the number of elements.

4.1.1.2 Sum Type

The type $A + B$ is a Sum of two types A and B that contains exactly one element of type A or B at any one time. This type is commonly referred to as tagged union or variant.

4.1.1.3 Unit Type

A Unit type defined as () is a special type that can be described in terms of Product type as a 0-tuple, i.e. an empty type that has exactly one value, hence its symbol ().

4.1.1.4 Pure Function

A function that is defined as $f: A \to B$ is a function called f, whose argument is of type A and its return type is B. In other words, f is a function from type A to B. Function $f2$ in the following example is a curried version of $f1$ and $f3$ is a partially applied function $f2$ – it is a function from A to a function from B to C. Note that all these forms of this function are algebraically equivalent and therefore can be used interchangeably.

$$f1: (A, B) \to C \approx f2: A \to B \to C \approx f3: A \to (B \to C)$$

4.1.1.5 Parametric Polymorphism

Parametric polymorphism can be used to make types or functions generic, meaning they can be described in terms of untyped element that can be replaced with element of any type, which has to be matching for all its occurrences.

For an example of a generic function let us look at $fst: (a, b) \to a$. The function fst takes a tuple of two elements of generic types a and b and returns the first one. An example of a generic type is illustrated in section 4.1.1.10 below.

4.1.1.6 Recursive Type

Recursive type is a type that is defined in terms of itself. An example of this type can be found section 4.1.1.11 below.

4.1.1.7 Primitive Types

The primitive types are:

Boolean Type

Boolean Type is defined as $Bool := True + False$

Integer Types

Primitive integer types are $UIntN$ and $IntN$, where N is size of the its internal representation i.e. 8, 16, 32 or 64 bits.

Floating Point Types

Two floating point types are $Float$ and $Double$.

String Type

String type is simply defined as $String$. The strings used in this project will be encoded in UTF-8 Unicode format (Radzivilovksy, et al., 2015).

4.1.1.8 Impure Function

To encapsulate functions that modify some global state or its argument while preserving its purity Haskell uses a monadic type IO (Peyton Jones, 2010). A pictorial representation of this type is $IO\ a := World \rightarrow (a, World)$.

4.1.1.9 Mutable Type

A mutable variable is formulated as $IORef\ a$. This kind of variable can only be used inside an IO monad.

4.1.1.10 Maybe Type

A generic type defined as $Maybe\ a := () + a$ is type called $Maybe$ with type parameter a that is sum of unit type $()$ and a. This type is useful for storing optional data or for returning values from impure functions that might fail for various reasons.

4.1.1.11 List Type

A classic example of recursive type is a polymorphic type $List$ defined as $List\ a := () + (a \times List\ a)$. A $List$ is either empty in which case it contains a Unit type, or it contains a Product of an element of type a and another $List$ of the same parametric type a.

4.1.1.12 Map Type

Map type is a type that stores value in a field associated with a key, which can be used to access this value. It is formally defined as $Map\ k\ a$, where k is the generic type of its keys and a is the generic type of its values.

4.1.2 Game Engine

The design and implementation of algebraic data type system has been followed by design and implementation of all the modular parts of the game engine. The modules are organised into subfolders to simplify orientation in the project.

4.1.2.1 Window

A window can be initialised with configurable properties such as its width, height style and number of bits per pixel and will be implemented using the Windows API. All the functions associated with window will be performing input or output operations. Window configuration is formally defined below. These definitions use the ADT types defined in section 4.1.1. The symbol ≡ is in here used to mean "is of the same type as".

$$WindowConfig := Width \times Height \times WindowStyle \times BitsPerPx$$

$$Width \equiv Heigh \equiv BitsPerPx := UInt$$

$$WindowStyle := Window + Fullscreen$$

The function that opens a window is defined as:

$$open: WindowConfig \rightarrow IO(Maybe\ Window)$$

4.1.2.2 Utilities

Windows API accepts "wide-character" strings instead of UTF-8 encoded Unicode string that are being used in this project as declared in section 4.1.1.7. Therefore a simple function that converts from UTF-8 to wide-character string was needed. This function was adpoted in more than one module and therefore it was placed inside Utilities module.

4.1.2.3 Renderer

Windows 8.1 updated Direct3D version 11.2 that is now part of its SDK (Walbourn, 2013) will be used to implement game engine's renderer, but also to compile HLSL shaders and load textures. The formal symbol for renderer will be $Renderer$. Similarly to $Window$, all the $Renderer$ operations perform input or output, which means they are also impure.

The shader that has been used for material implementation is identical for both FP and IP versions and therefore plays no role in this research.

4.1.2.4 Timer

A simple timer needs to be implemented to keep track of elapsed time, which is typically needed for calculations of animations, physics simulation and any other time dependent game logic. Timer and its operations are defined as:

$$Timer := DeltaTimeMs \times TimeMs$$

$$DeltaTimeMs \equiv TimeMs := Double$$

$$initTimer: () \rightarrow IO(IORef\ Timer)$$

$$updateTimer: IORef\ Timer \rightarrow IO()$$

4.1.2.5 Engine

The engine and its functions tie the modules together, it is responsible for opening a window, initialising renderer, and then in infinite loop processing input messages, updating timer, and updating and rendering game actors (once the actor system is designed and implemented), until the game is terminated. To simplify the engine's interface, all these tasks can be launched by a single impure function described as:

$$run: IORefEngine \rightarrow WindowConfig \rightarrow IO()$$

4.1.2.6 Window Revisited – Game Input

The window procedure callback function is responsible for processing game input that gets updated in every frame. Game input unifies state of all the input devices into a single data structure and is formally defined as:

$$GameInput := Map\ Input\ Boolean\ \times Mouse\ \times Text$$

$$Input := KeyInput\ \times MouseInput$$

$$KeyInput := Backspace + Tab + Return + Shift + Ctrl\ \ldots$$

$$MouseInput := LeftButton + RightButton + MiddleButton + XButton1 + XButton2$$

$$Mouse := X \times Y \times Delta$$

$$X \equiv Y \equiv Delta := UInt16$$

$$Text := UInt32$$

The $GameInput$ maps all the keys and mouse buttons to a $Bool$ value (section 4.1.1.7), that is $True$ if the key or button is pressed and $False$ otherwise. Because of the design of Windows API's callback procedure function, $Window$'s $open$ function now requires a reference to $Engine$ to feed the $GameInput$ back into $Engine$, making the new function's signature:

$$open: IORef\ Engine \rightarrow WindowConfig \rightarrow IO(Maybe\ Window)$$

4.1.2.7 Maths and Geometry

Vector

There need to be at least three types of vector: two-dimensional, three-dimensional and four-dimensional with some usual functions such as calculation of cross product, dot product and normalised vector implemented. All the vectors will be parameterised with the type of its scalar components, so that its internal representation can be easily changed. The vector types can be used to represent vectors, points' position and Euler angles in respective dimensions. Vectors will be formally referred to as $Vec2\ a$, $Vec3\ a$ and $Vec4\ a$, where:

$$FVec2 := Vec2\ Float$$

$$FVec3 := Vec3\ Float$$

$$FVec4 := Vec4\ Float$$

Quaternion

Quaternion and its common operations are to be implemented to avoid the common issues of representing rotation as Euclidean vector such gimbal lock. Quaternion will

also be parameterised with its scalar component type and it will be symbolised by $Quat\ a$, where $FQuat := Quat\ Float$.

Matrix

Four by four matrix with its associated operations such as matrix multiplication, inverse and determinant is to be implemented for graphics calculations and 3D model transformations. Matrix formal symbol will be $Mat4x4$.

Plane

A plane representation in form of general plane equation is to be implemented to simplify definition of camera's frustum.

Camera's Frustum

Camera's frustum is used in rendering process to calculate camera's position and projection. It consists of six "clipping" planes: near, far, top, right, bottom and left and defines properties such as field of view, aspect ratio and near and far clipping plane distance from the camera.

Colour

A colour is represented by four values, each for one of its channels: red, green, blue and alpha or transparency.

4.1.2.8 Model and Material

In a 3D game engine, an actor is not defined just by its functionality, but also by its model and material. A model can be a built from built-in primitive geometry such as cube or loaded from a file. Properties of a material are driven by the pixel and vertex shaders it is using. The definition of model and material is formally described as:

$$ModelDef := BuiltInModelDef + LoadedModelDef$$

$$BuiltInModelDef := BuiltInModelType \times Dimensions$$

$$BuiltInModelType := Cube + Sphere$$

$$Dimensions := FVec3$$

$$LoadedModelDef := Filename \times Scale$$

$$Filename := String$$

$$Scale := Float$$

$$MaterialDef := DiffuseTexture \times SpecularTexture \times BumpTexture \times ParallaxTexture \times EnvMapTexture \times TextureRepeat$$

$$DiffuseTexture \equiv SpecularTexture \equiv BumpTexture \equiv ParallaxTexture \equiv EnvMapTexture := String$$

$$TextureRepeat := FVec2$$

4.1.2.9 Resources

To avoid unnecessary copying of resources, $Resources$ type defined below can be used to store references to them. Note that all the resources values are wrapped inside $Maybe$ type, because all these values are loaded obtained from IO functions, which can fail to load or initialise the resources for multiple reasons.

$$\begin{aligned} Resources := & \; Map\ LoadedModelDef\ (Maybe\ Model) \\ & \times Map\ BuiltInModelDef\ (Maybe\ Model) \\ & \times Map\ MaterialDef\ (Maybe\ Material) \\ & \times Map\ String\ (Maybe\ Texture) \end{aligned}$$

4.1.2.10 Actor System

An introduction to a typical FP approach to implement game engine's actor system called Functional Reactive Programming has been given in Literature Review section 2.3.1.2 and its design for this project is outlined below.

Functional Reactive Programming

The way Yampa implements FRP was a main source of inspiration for this project. It makes time omnipresent through use of signals. Signal is defined as a function from $Time$ to a generic value a, where $Time$ is a continuous real number:

$$Signal\ a := Time \rightarrow a$$

Signals are being composed together via signal functions that transform one signal into another:

$$SF\ a\ b := Signal\ a \rightarrow Signal\ b.$$

Yampa also provides a set of "combinators" for composition of signal functions. To "lift" an ordinary pure function to a signal function there is a function called arr (lifting a function is a very common concept in FP and it typically used to transform a pure function to be applicable to some encapsulated type – in here signal). When given a function $f: a \rightarrow b$, it returns a signal function $SF\ a\ b$. It is defined as:

$$arr: (a \rightarrow b) \rightarrow SF\ a\ b$$

The purpose of the other combinator functions is to compose signal functions together with functions such as (the functions are written in their curried form which is formulated in section 4.1.1.4):

$$(>>>): SF\ a\ b \rightarrow SF\ b\ c \rightarrow SF\ a\ c$$

$$(<<<): SF\ b\ c \rightarrow SF\ a\ b \rightarrow SF\ a\ c$$

The symbols $(>>>)$ and $(<<<)$ in the definitions above are the functions' names. On top of that Yampa uses Arrow notation for composing signals with signal functions. The diagram of Yampa signal functions (Meisinger, 2011) in Figure 4.2 below helps to visualise how some of the signals, signal functions and their compositions work.

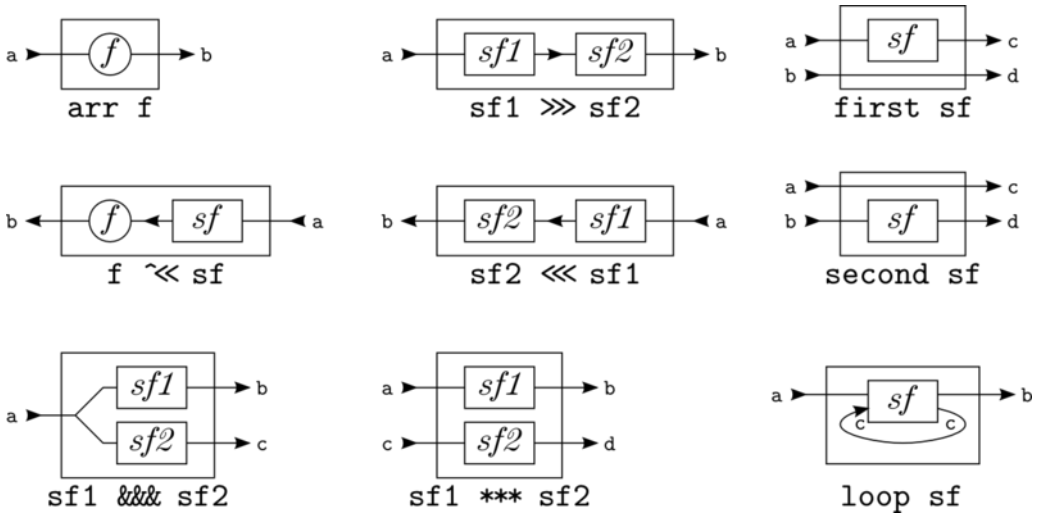


Figure 4.2 Yampa signal functions (Meisinger, 2011)

Actor Definition

Following the process of Data-Oriented design, the actor system has been formally defined and implemented. In this implementation, the definition of actor will be formulated as combination of actor type, starting state, its signal function from actor input to actor output and finally a recursive list of its children, formally written as:

$$ActorDef := ActorTypeDef \times ActorStartingState \times SF\ ActorInput\ ActorOutput \times List\ ActorDef$$

Actor type is either model or camera:

$$ActorTypeDef := ActorModelDef + ActorCameraDef$$

Actor model is combination of its model and material, both of which are defined in the following section 4.1.2.8:

$$ActorModelDef := ModelDef \times MaterialDef$$

The definition of camera can be found in section 4.1.2.11. The rest of the actor definition follows in similar fashion:

$$ActorStartingState \equiv ActorState := Position \times Velocity \times Scale \times Rotation \times ModelRotation$$

$$Position \equiv Velocity \equiv Scale := FVec3$$

$$Rotation \equiv ModelRotation := FQuat$$

$$ParentTransform := Mat4x4$$

$$ActorInput := GameInput \times ActorState$$

$$ActorOutput := ActorState$$

Initialised Actor

After an actor is initialised from its $ActorDef$ definition, it should then take on the following form:

$$Actor := ActorState \times SF\ ActorInput\ ActorOutput \times RenderFn \times List\ Actor$$

$$RenderFn := IORef\ Renderer \rightarrow ActorState \rightarrow ParentTransform \rightarrow IO()$$

$$ParentTransform := Mat4x4$$

The first thing to notice here is that the actor has no apparent reference to its model or material. That is because the rendering function $RenderFn$ will be initialised to capture reference to any resources the actor uses instead and therefore $Actor$ can have uniform type regardless if it uses a model, a material or anything else. The users of the engine can then customise the rendering function to any kind of functionality without having to rewrite any part of the engine.

4.1.2.11 Camera

Camera has been designed as a type of an actor, to be the part of the actor hierarchy and provide a simple way to manipulate camera and customise its behaviour. The renderer function needs to know about the camera's properties, because they are used in the shader to calculate world-view-projection matrix and the specular component. For this purpose the actor's rendering function may be used. The definition of an actor camera is formally described as:

$$ActorCameraDef := NearClipDist \times FarClipDist \times InitCamRenderFn$$

$$NearClipDist := Float$$

$$FarClipDist := Float$$

$$InitCamRenderFn: ActorCamera \rightarrow WindowConfig \rightarrow RenderFn$$

Once the game engine initialises the camera it takes on the same format as actors, which is described in the previous section.

4.1.2.12 Engine Revisited

Now that the actor system has been developed, the rest of the game engine's functionality can be implemented. The engine's run function can already open $Window$, initialise $Renderer$ and $Timer$, and then the main loop processes input messages and updates $Timer$. To make use of the other modules, further it should initialise $Resources$, initialise actors defined as $List\ ActorDef$ into $List\ Actor$, and then in the main loop call the signal functions of the actors before rendering them. The revisited run function is described as:

$$run: IORefEngine \rightarrow List\ ActorDef \rightarrow WindowConfig \rightarrow IO()$$

4.2 Implementation of Functional Programming Version

Milewski (2011) has shown the many similarities between Haskell and C++ template metaprogramming and how many ideas from Haskell can be implemented in C++ using templates, even though the syntax to do so is not as clear and concise as in Haskell. The reason for this is that while Haskell was designed with parametric polymorphism from the beginning (Hudak, et al., 2007), in C++ the usefulness of templates has been discovered by accident as explained in section 3.3.1.3.

4.2.1 Algebraic Data Types

4.2.1.1 Product Type

This type is used for composite types that need to contain elements of multiple types at the same time such as the two-dimensional vector shown in Listing 4.1. Its implementation in C++ is very straight-forward, it is a structure (or a class) containing a member variable for each operand of the Product type, in here `x` and `y` both of type `float`.

```
1. struct Vector2
2. {
3.     float x, y;
4. };
```

Listing 4.1 Two-dimensional vector is a Product type

4.2.1.2 Sum Type

For Sum types without arguments and nested data such as $EngineState$ shown below simple `enum` can be used.

$$EngineState := Initialised + Running + Terminated$$

For other Sum types, the simplest way to define Sum type in C++ is by using `union`, however it is not the most convenient construct to use, because each `union` type also has to manually implement a way to check which operand does the Sum type contain. Another limitation of `union` is that it can only contain plain-old-data (POD) types, but for the purpose of this project, that is not important because it has been decided in section 3.4.2 that object-oriented classes will not be used.

An alternative solution would be to use the C++ template power to define a better Sum type, such as the Variant class implemented in Boost library that offers type-safety through use of visitor pattern (Polukhin, 2013). However, for a reason that such an implementation is non-trivial (Boost version 1.57.0 Variant library contains over 6000 lines of code) and for the purpose of this project it would outweigh the benefit of having such a type at hand, it has been decided that a plain `union` type will be used instead.

To make working with union simpler, it would be handy to have unique type identification for every type at runtime, but because RTTI is disabled, this information is not available. For that reason a helper type `TypeId` (first line of Listing 4.2) and associated template function `typeId` (line 3 onwards in Listing 4.2) have been implemented. This function returns a unique `TypeId` for any template parameter. Under the hood, `TypeId` is a pointer to a static variable that the compiler generates for every type that uses this function.

```
1. typedef void* TypeId;
2.
3. template<typename A>
4. TypeId typeId( )
5. {
6.     static A* typeUniqueMarker = NULL;
7.     return &typeUniqueMarker;
8. }
```

Listing 4.2 Type identification without RTTI

In the next Listing 4.3 is an example of Sum type called `ModelDef`, this Sum type's operands are `BuildInModelDef` and `LoadedModelDef` (lines 10 and 11). The literal meaning of this Sum type is: `ModelDef` can be either `BuildInModelDef` or `LoadedModelDef`. The lines 13 to 17 consist of a boilerplate code that is the same for all the Sum types. The factory functions on lines 18 and 19, whose purpose is to construct a `ModelDef` instance, are similar for all the Sum types.

```
1.  struct BuiltInModelDef;
2.  struct LoadedModelDef;
3.  struct ModelDef
4.  {
5.  private:
6.      TypeId _typeId;
7.  public:
8.      union
9.      {
10.         BuiltInModelDef builtIn;
11.         LoadedModelDef loaded;
12.     };
13.     template<typename A>
14.     bool is( ) const
15.     {
16.         return ( _typeId == typeId<A>( ) );
17.     }
18.     friend ModelDef builtInModelDef( BuiltInModelDef&& m );
19.     friend ModelDef loadedModelDef( LoadedModelDef&& m );
20. };
```

Listing 4.3 Example of Sum type

Implementation of one of the factory functions can be seen in Listing 4.4. It uses C++11 move semantics for its argument for best efficiency, instantiates a new `ModelDef` object, sets it to be `BuildInModelDef` and sets its `TypeId` using the previously described function `typeId`. Thanks to C++ feature called Return value optimization (abbr. RVO), the object can be returned by value without creating any unnecessary copies (Josuttis, 2012).

```
1. ModelDef builtInModelDef( BuiltInModelDef&& m )
2. {
3.     ModelDef model;
4.     model.builtIn = m;
5.     model._typeId = typeId<BuiltInModelDef>( );
6.     return model;
7. }
```

Listing 4.4 Example of a Sum type's factory function

4.2.1.3 Unit Type

Unit type is very simple to implement in C++ as can be seen in Listing 4.5, however, it's not been used in this project, because for the types where the Unit type could potentially be useful, there was a simpler implementation possible, which is explained in section 4.2.1.10 below.

```
struct Unit { };
```

Listing 4.5 Unit type implementation

4.2.1.4 Pure Function

There are multiple ways to define function $f: A \to B$ in C++ that are usually useful for different situations. Example of every style together with syntax for its function pointer follows. Listing 4.6 lines 5 through 8 shows function `f_1` written in the most commonly used C style function definition and line 9 is a function pointer named `fPtr_1` to this function.

```
1. struct A
2. { };
3. struct B
4. { };
5. B f_1( A )
6. {
7.     return B{ };
8. }
9. B( *fPtr_1 )( A ) = f_1;
```

Listing 4.6 C style function

Listing 4.7 is using a technique of overloading function application operator to create callable object named `f_2`. The advantage of this definition over C style function definition is that callable object may be simpler to pass into function because function pointer `fPtr_2` on line 17 has simpler syntax than that of C style function.

```
10. struct f_2
11. {
12.     B operator () ( A )
13.     {
14.         return B{ };
15.     }
16. };
17. f_2 fPtr_2;
```

Listing 4.7 Callable object

C++11 standard introduced alternative way to C style function definition with trailing return type which can be seen in Listing 4.8. The difference between the two is that this new form has its arguments defined before its return type (notice the similarity to Haskell function definition that for this function would be `f_3 :: A -> B`). The reason for this is that some generic programming techniques were not possible with the original C style definition. An example of such a technique can be found in section 4.2.1.10 in function defined in Listing 4.16 that deduces function return type from its arguments.

```
18. auto f_3( A ) -> B
19. {
20.     return B{ };
21. };
22. B( *fPtr_3 )( A ) = f_3;
```

Listing 4.8 C++11 function definition with trailing return type

Listing 4.9 shows anonymous function (more commonly referred to as lambda function) definition, that was first introduced in C++11, being assigned to function pointer `fPtr_4`. This notation is just a syntactic sugar for callable object which it uses under the hood. The improvement over the callable object is that lambda functions can be defined locally inside functions scope and function arguments unlike callable objects, and because lambdas can also automatically deduce return type, in most cases its specification is optional.

```
23. auto fPtr_4 = []( A ) -> B
24. {
25.     return B{ };
26. };
```

Listing 4.9 C++11 lambda function

Listing 4.10 demonstrates a very similar definition to the previous one that uses STL `std::function` wrapper. The main difference between them is that type of function pointer `fPtr_4` is closure and `fPtr_5` is `std::function`, which can sometimes play a role when, for example, the function is being passed into another function as its argument. Some argument types may accept one form but not the other.

```
auto fPtr_5 = std::function < B( A ) >
{
    []( A )
    {
        return B{ };
    }
};
```

Listing 4.10 STL function wrapper

4.2.1.5 Parametric Polymorphism

As shown in Listing 4.11 generic function can be defined by application of template metaprogramming. An example of a generic type can be seen in section 4.2.1.10 below.

```
template<A, B>
A fst( std::tuple<A, B> pair )
{
    return std::get<0>( pair );
}
```

Listing 4.11 Generic function in C++

4.2.1.6 Recursive Type

List previously defined as $List\ a := () + (a \times List\ a)$ is implemented in Listing 4.12. The reason that there is no unit type in the definition is because it uses a simplification that is illustrated in the following section 4.2.1.10.

```
template<typename A>
struct List
{
    struct Cons
    {
        A head;
        List<A> tail;
    };
    Cons* cons;
};
```

Listing 4.12 List is a recursive type

4.2.1.7 Primitive Types

All the primitive types have been defined in the precompiled header file using built-in primitive types.

4.2.1.8 Impure Function

Impure functions are in Haskell implemented using monads. There are some experimental implementations of a monad in C++, however, the concept is not yet fully possible to implement in C++. Milewski (2014) discussed the possibility of monad being implemented in the upcoming standard C++17.

A simple wrapper type for impure functions defined in Listing 4.13 does not offer a great benefit, on the contrary it only increases the code complexity, because in C++ the rules that the IO type has in Haskell cannot be enforced, but also because in impure functions that do not return any value (i.e. their return type would be `IO<void>`) this type information is lost. Therefore it has been decided to annotate the impure functions in C++ by appending their name with a string "_IO" instead, which serves as a reminder to programmer that the function might have side-effects.

```
1. template<typename A>
2. struct IO
3. {
4.     typedef std::function<A( )> type;
5. };
```

Listing 4.13 Naive C++ implementation of IO type

4.2.1.9 Mutable Type

All C++ variables are mutable unless they are marked as `const`. Furthermore, all the functions that accept a mutable variable as their argument must be defined as impure. However, because it is not most convenient to only work with `const` variables in C++, mutable variables can still safely be used inside pure functions if they do not escape their scope.

4.2.1.10 Maybe Type

For the Maybe type defined as $Maybe\ a := () + a$ whose literal C++ implementation is shown in Listing 4.14 below, there is a simpler implementation possible. If Unit is defined as `nullptr`, the Maybe type can be expressed as a pointer to the operand data, which has optional value in C++ (Listing 4.15). The very same method has also been used in other types that contain optional value, such as the example of recursive type in the section 4.2.1.6 above.

```
1. template<typename A>
2. struct Maybe
3. {
4.     union
5.     {
6.         Unit nothing;
7.         A something;
8.     };
9. };
```

Listing 4.14 Naive implementation of Maybe type

```
1. template<typename A>
2. struct Maybe
3. {
4.     A* something;
5. };
```

Listing 4.15 Simpler implementation of Maybe type

To encapsulate the `Maybe`'s inner implementation and to be able to conveniently construct an instance of type `Maybe`, I have implemented two factory functions, called `just` and `nothing`, that construct `Maybe` with initialised data and empty data respectively.

I have also implemented function `ifThenElse` using visitor pattern that takes an instance of `Maybe` and two functions. If the `Maybe` object is not empty then the first function is called with the object's inner data and its result returned by this function, otherwise the second function is called and its result returned instead. Implementation of the function can be seen in Listing 4.16. A careful reader will notice that instead of return type on line 3, the function definition contains `decltype( ifJust( *maybe._a ) )`. This expression is being used to deduce the function return type from the return type of its second argument. The assertion on line 5 helps to improve safety of this function by checking that the return type of the first and second function is the same. Another constraint that was not possible to express in C++ without convoluted type trait trickery is that the first function's argument type should be the same as the template parameter of the Maybe object.

```
1. template<typename A, typename B, typename C>
2. auto ifThenElse( const Maybe<A>& maybe, B ifJust, C ifNothing ) ->
3. decltype( ifJust( *maybe._a ) )
4. {
5.     static_assert( std::is_same < decltype( ifJust( *maybe._a ) ),
6.         decltype( ifNothing( ) ) > ::value );
7.     if ( maybe._a == nullptr )
8.     {
9.         return ifNothing( );
10.    }
11.    return ifJust( *maybe._a );
12.}
```

Listing 4.16 Maybe's visitor function

One might ask why not use a simpler declaration such as the one in Listing 4.17. The reason is that such an interface does not accept closures, which is a serious limitation.

```
1. template<typename A, typename B>
2. B ifThenElse( const Maybe<A>& maybe, B( *ifJust )( const A& a ), B(
   *ifNothing )( ) );
```

Listing 4.17 Maybe's visitor function alternative implementation

This issue could be elevated by changing the type of function arguments to use STL function wrapper as can be seen in Listing 4.18, however this style of declaration comes with a different issue, that is C++ cannot deduce type from `std::function` template parameters and therefore they would have to be specified every time this function is being used.

```
1. template<typename A, typename B>
2. B ifThenElse( const Maybe<A>& maybe, std::function<B( A )> ifJust,
   std::function<B( A )> ifNothing );
```

Listing 4.18 Another Maybe's visitor function alternative implementation

4.2.1.11 List and Map

For simplification STL `std::vector` has been used for *List* type and `std::map` for *Map* type, instead of an actual FP implementation. This is a limitation that is considered in the section 6.3, which talks about possible further work.

4.2.2 Game Engine

4.2.2.1 Window and Game Input

The `open_IO` function parameter is an instance of `WindowConfig`. Before it attempts to open a window, it checks whether another window is already open. It returns an instance of `Window` wrapped inside `Maybe` type, which is empty if something in the process fails, for example if any of the Windows API calls fail for whatever reason or if the window is already open. This module also provides functions for obtaining the desktop's window configuration, switching to fullscreen window and capturing and releasing mouse, which is typically handy for making capturing all the mouse messages even when the mouse cursor moves out of window.

4.2.2.2 Utilities

Some utilities that are used consistently throughout the whole project have been decided to be declared inside precompiled header file. The other less often used functions that are used in multiple places throughout the engine but do not fall under category of any of the modules, have been decided to be placed inside Utilities module.

There are multiple useful utilities defined in the precompiled header file, such as various logging macros with different levels of criticality, a macro for operator `new`, `delete`, `delete []` for array de-allocation, and `Release` method that enhances and simplifies these often used actions. It also defines some constants that are generally useful in a game engine such as π constant and factors for conversion from degrees to radians and other way round.

4.2.2.3 Renderer

As mentioned earlier, `Renderer` has been implemented using Direct3D version 11.2. It uses double buffered swap chain for rendering and provides rendering functions for 3D models by its vertices and indices. Index is simply defined as `UInt32`, while vertex is a Product type of its position, colour, texture coordinates, and normal, tangent, and bi-normal vector. Renderer also provides function `preRender_IO` for clearing the render

target view and depth stencil that is typically called every time before it renders anything on the screen (otherwise artefacts from the previous render loop iteration may still be visible) and function `present_IO` that swaps the renderer's double buffer and presents it to the screen.

4.2.2.4 Timer

Dawson (2012) has shown that to achieve the best precision, timer should store the elapsed time in a `double` variable that starts counting from 2^{32} (Dawson, 2012). The timer associated update function is also impure, because it obtains time through Windows API.

4.2.2.5 Engine

Engine can have three possible states: `Initialised`, `Running` and `Terminated`. The function `init` can be used to initialise an instance of Engine in `Initialised` state. The engine is then launched through its function `run_IO`, which attempts to open a window. It then uses the visitor function `ifThenElse` together with lambda expressions to define what happens next. If the `Maybe<Window>` is not empty, it then goes on to initialise renderer. In the same manner, the `ifThenElse` function is used again to check that the renderer has been successfully initialised, in which case the engine's state is set to `Running` and the main loop starts getting executed.

Unfortunately, it is not possible to use a recursive function for the main loop, because C++ cannot optimise it and so any attempt to use recursion for the main loop results in stack overflow. The reason that in Haskell it is possible to use recursion for main loop without running into the same issue is because Haskell uses lazy evaluation, which mitigates most of the problems associated with recursive calls.

The game engine's main loop then processes input messages and updates timer, until the engine gets terminated, in which case the window procedure function changes the engine's state to `Terminated`, which is the condition for the main loop to end.

4.2.2.6 Window Revisited – Game Input

The `open_IO` function now requires another argument which is a mutable reference to `Engine`, to be able to pass input from the window procedure function back to `Engine` through use of `lpParam` argument of the Windows API `CreateWindow` function. The window procedure function `windowProc_IO` handles the processing of input (keyboard, mouse and other special messages) that is being fed back to `Engine`.

4.2.2.7 Maths and Geometry

All the previously discussed concepts have been implemented in data-oriented style and all their associated operations implemented as pure functions.

4.2.2.8 Model and Material

Autodesk's FBX SDK has been used for loading 3D models from FBX files and Microsoft's DirectXTex library for loading textures from files.

4.2.2.9 Resources

Resources are implemented as a collection of map data structures from model and material definitions to the actual models and materials.

4.2.2.10 Actor System

To develop the actor system Functional Reactive Programming system had to be implemented first.

Functional Reactive Programming

The concepts of signals and signal functions described in section 4.1.2.10 have been implemented literary as they are defined. Signal is a generic structure parameterised over its value with a member function pointer to a function from time to value as can be seen in Listing 4.19.

```
1. template<typename A>
2. struct S
3. {
4.     std::function<A( const float )> f;
5. };
```

Listing 4.19 Signal implementation

Similarly, the signal function is a generic structure parameterised over its argument (`A`) and return type (`B`) with a member function pointer to a function from signal of `A` to signal of `B`.

```
1. template<typename A, typename B>
2. struct SF
3. {
4.     std::function<S<B>( const S<A>& a )> f;
5. };
```

Listing 4.20 Signal function implementation

To keep the way users can compose signals and signal functions uniform, the operator "<" has been overloaded for multiple arguments. For example Listing 4.21 lines 6 through 9 shows a function that can be used to apply a signal to signal function and lines 10 through 18 shows a function that can be used to compose two signal functions. Notice that the second function's inner closure captures the signal functions by value which creates a new copy of both. The reason for this is when capture by reference is used instead the references get lost which results in crash of the application. There are many other combinator functions that had to take on the same approach. This has proved to be the biggest source of inefficiency of the FRP implementation, as argued in the following chapter.

```
1. template<typename A, typename B>
2. struct SF
3. {
4.     std::function<S<B>( const S<A>& a )> f;
5.
6.     S<B> operator < ( const S<A>& a ) const
7.     {
8.         return f( a );
9.     }
10.    SF<C, B> operator < ( SF<C, A>& sf ) const
11.    {
12.        return SF < A, C > {
13.            [sf, *this]( const S<A>& a ) -> S < C >
14.            {
15.                return snd.f( fst.f( a ) );
16.            }
17.        };
18.    }
19.};
```

Listing 4.21 Application of signal to a signal function and composition of two signal functions

Actors

The Actor module implements a default initialisation function of rendering function called `initActorRenderFunction_IO`. This initialisation function takes `Renderer`, `Resources` and `ActorDef` as its arguments to use `Resources` to load the actor's model and material based on its `ActorDef`. The instance of `Renderer` is needed for initialisation of the model and material. In case the `ActorDef` contains `ActorModelDef`, this function returns default rendering function, which uses closure to capture reference to the actor's resources. However, if the `ActorDef` contains `ActorCameraDef` instead, this function returns the camera's rendering function.

The actor input and actor output is implemented as simply as it was defined in section 4.1.2.10, which can be seen in Listing 4.22 below.

```
1. struct ActorInput
2. {
3.     GameInput gameInput;
4.     ActorState state;
5. };
6. struct ActorOutput
7. {
8.     ActorState state;
9. };
```

Listing 4.22 Actor input and output

4.2.2.11 Camera

The user of the game engine can customise the camera's behaviour through its placement in the actor hierarchy and its *InitCameraRenderFn* function. This is

implemented as a function that accepts `ActorCameraDef` and `WindowConfig` as its arguments that can be used to calculate the camera's properties. It should then return a rendering function. Furthermore, the rendering function when written as closure over the camera's properties can be used to feed this information to the renderer.

4.2.2.12 Engine Revisited

The engine's `run_IO` function now needs to incorporate rest of the modules to make the main loop complete. Before it enters the main loop, the actor hierarchy that is now one of its arguments needs to be initialised. This is done through a recursive function called `initActors_IO` that loops through the actor hierarchy. This function uses the function `initActorRenderFunction_IO` previously described in section 4.2.2.10 above.

After that the function can enter the main loop, in which three more function calls had to be added. First one is a call to `preRender_IO` that clears the renderer target and depth stencil. Second is a call to function `renderActors_IO`, which is another recursive function looping through the actor hierarchy. This is the function that activates the FRP system, by applying an instance of `ActorInput` into actor's signal function, which returns `ActorOutput` that contains a new actor's state. Finally the main loop calls renderer function `present_IO` to draw the rendered image on the screen.

4.3 Design and Implementation of Imperative Programming Version

The design and implementation of the game engine in imperative programming style follows much more common architecture and therefore this section is not as detailed as the previous sections that focused on the functional programming version.

Where the engines differ the most is the actor system, that for the Imperative Programming version is component based. Class diagram of this system is shown in Figure 4.3 below. Component based actor system is designed around mutation of the state of actors and their components, that is controlled through overriding of the virtual `vUpdate` method. The relation between actor and components is one-to-many.

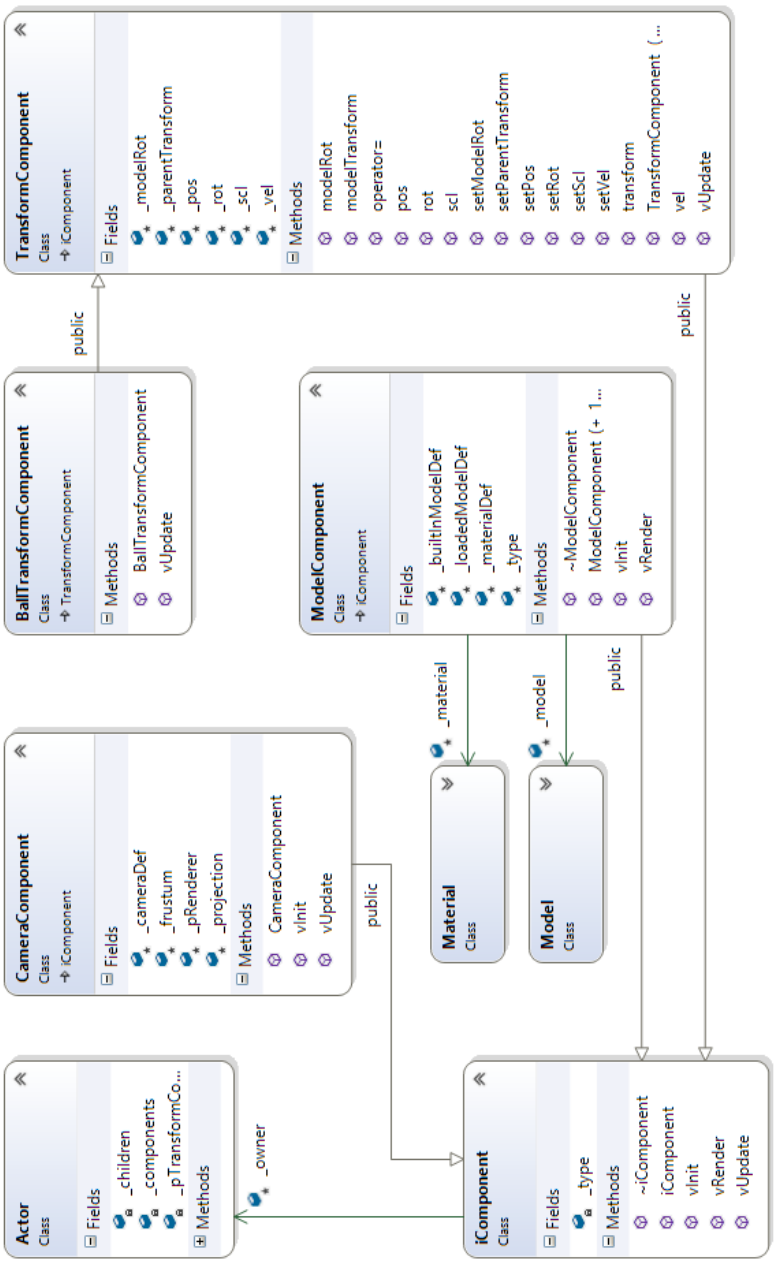


Figure 4.3 Component based actor class diagram

5 Testing and Evaluation

The test phase involved putting both of the implemented versions under evaluation, both practical and theoretical. The practical part involved further development that was required to measure performance of the engine and to further evaluate advantages and disadvantages of FP.

The analysis involved comparison of the measured data, and a discussion of its results together with the non-quantifiable results followed. As most of the qualities that FP promises are not measurable, the conclusion will most likely serve as an indicator of viability of using FP concepts in C++, which further work can draw upon, rather than arriving at a final verdict about which approach is better.

5.1 Demonstration of the Game Engines

To put both engine versions to a practical test a demonstration application has been designed and developed using both of these versions to compare the ease of implementation, readability, and to perform profiling of the application to measure the game engines' efficiency. The specification of it follows. Because one of the hypotheses was that functional programming can speed-up prototyping process and another was that functional programming improves code readability and maintainability the following description is thorough to support conclusion on these topics.

5.1.1 Specifications

The application simulates a ball controlled by the player viewed from first person perspective. This means that the camera has to follow the movement of the ball at certain offset. The possible manoeuvres of the ball are described in Table 5.1 below that lists application controls.

Table 5.1 Demonstration application controls

Command	**Action**
W key	Move forward
S key	Move backward
A key	Turn left
D key	Turn right
Q key	Strafe left
E key	Strafe right
Spacebar	Jump

5.1.1.1 Physics

The ball should maintain its velocity, but it should get damped over time. It should only be possible to move forward, backward and strafe left and right when the ball is on the

ground. When the ball hits the ground after it jumps, it should rebound at half of its velocity. The formal description of the ball's physics simulation and its properties follows.

Acceleration: 0.001 $\frac{\mathrm{m}}{\mathrm{ms}^2}$

Jump velocity: 0.01 $\frac{\mathrm{m}}{\mathrm{ms}}$

Angular velocity: 0.003 $\frac{\mathrm{rad}}{\mathrm{ms}}$

Minimum position on X axis: 0.45 m

Gravitation: 9.81 $\frac{\mathrm{m}}{\mathrm{s}^2}$ = 0.00000981 $\frac{\mathrm{m}}{\mathrm{ms}^2}$ (gravity of Earth)

Coefficient of restitution: -0.5

Maximum velocity: 0.01 $\frac{\mathrm{m}}{\mathrm{ms}}$

5.1.1.2 Actor Hierarchy

The simple actor hierarchy can be described as follows:

- Ball
  - Camera
- Ground

5.1.1.3 Game Art

The 3D models and textures in Table 5.2 below were obtained for the demonstration from sources listed in Appendix A.

Table 5.2 Game art list

Actor	**3D Model**	**Textures**
Ball	basketball [loaded]	basketball [diffuse, bump]
Ground	cube [pre-built]	wooden tiles [diffuse, bump]

5.1.2 Implementation of Functional Programming Version

A sample of actor definition in FP version can be seen in Listing 5.1 below. It is using C++11 uniform initializer lists.

```
1. auto actors = {
2.     actorModelDef( {
3.         loadedModelDef( { // model
4.             "basketball.fbx", // filename
5.             0.02f // scale
6.         } ),
7.         { // material
8.             "basketball-diffuse.jpg", // diffuseTextureFilename
9.             "", // specularTextureFilename
10.            "basketball-bump.jpg", // bumpTextureFilename
11.            "", // parallaxTextureFilename
12.            "" // evnMapTextureFilename
13.        }
14.    } ),
15.    { // startingState
16.        { 0.0f, 0.45f, 0.0f }, // pos
17.        { 0.0f, 0.0f, 0.0f }, // vel
18.        { 1.0f, 1.0f, 1.0f }, // scl
19.        FQuat::identity, // rot
20.        FQuat::identity // modelRot
21.    },
22.    ball( ), // sf
23.    { // children
24.        {
25.            actorCameraDef( {
26.                0.001f, // nearClipDist
27.                1000.0f, // farClipDist
28.                initCameraRenderFn( ) // render
29.            } ),
30.            { // startingState
31.                { 0.0f, 2.0f, -7.0f }, // pos
32.                { 0.0f, 0.0f, 0.0f }, // vel
33.                { 1.0f, 1.0f, 1.0f }, // scl
34.                FQuat::identity, // rot
35.                FQuat::identity // modelRot
36.            },
37.            staticActor( ), // sf
38.            { } // children
39.        }
40.    }
41. };
```

Listing 5.1 Actor definition in FP version

Line 22 in Listing 5.1 above defines the signal function of the ball object.

```
1. GameInput getGameInput( const ActorInput& input )
2. { return input.gameInput; }
```

Listing 5.2 Getter of game input

```
SF<ActorInput, ActorOutput> ball( )
{
    return SF < ActorInput, ActorOutput >
    {
        []( const S<ActorInput>& input ) -> S < ActorOutput >
        {
            static const float acceleration = 0.001f;
            static const float jumpSpeed = 0.01f;
            static const float rotSpeed = 0.003f;
            static const float minPosX = 0.45f;
            static const float gravity = -0.00000981f;
            // coefficient of restitution
            static const float cor = -0.5f;
            auto state = arr( getState ) < input;
            auto gi = arr( getGameInput ) < input;
            // stop from going under minPosX
            S<bool> onTheGround =
                arrAlt<float, bool>( eq( minPosX ) )
                < arr( getY ) < arr( getPos ) < state;
            // controls
            // accelerate forward
            S<float> accZForward =
                arrAlt<bool, float>( ifElse( acceleration, 0.0f ) )
                < and( onTheGround )
                < arrAlt<GameInput, bool>( getInputState( Key::W ) )
                < gi;
            // accelerate backward
            S<float> accZBackward =
                arrAlt<bool, float>( ifElse( -acceleration, 0.0f ) )
                < and( onTheGround )
                < arrAlt<GameInput, bool>( getInputState( Key::S ) )
                < gi;
            // anti-clockwise angular velocity
            S<float> angVelZLeft =
                arrAlt<bool, float>( ifElse( -rotSpeed, 0.0f ) )
                < arrAlt<GameInput, bool>( getInputState( Key::A ) )
                < gi;
            // clockwise angular velocity
            S<float> angVelZRight =
                arrAlt<bool, float>( ifElse( rotSpeed, 0.0f ) )
                < arrAlt<GameInput, bool>( getInputState( Key::D ) )
                < gi;
            // jump
            S<float> velYUp =
                arrAlt<bool, float>( ifElse( jumpSpeed, 0.0f ) )
                < and( onTheGround )
                < arrAlt<GameInput, bool>( getInputState(
                Key::Space ) ) < gi;


            /*
            ...
            */


```

Listing 5.3 Signal function of ball 1 of 2

```
56.             // rotation
57.             S<FQuat> rot = mul( arr( getRot ) < state )
58.                 < arr( eulerRadToQuat<float> )
59.                 < arrAlt<float, FVec3>( conFVec3fromY( ) )
60.                 < integral<float>( ) < add( angVelZLeft )
61.                 < angVelZRight;
62.             // acceleration
63.             S<FVec3> accZ = arrAlt<float, FVec3>( conFVec3fromZ( ) )
64.                 < add( accZForward ) < accZBackward;
65.             // new actor state
66.             auto newState = setRot( rot ) < state;
67.             // model rotation
68.             auto modelRot = mul( arr( getModelRot ) < newState )
69.                 < arr( rollBall ) < integral<FVec3>( ) < vel;
70.             // construct new actor output
71.             return arr( conActorOutput ) < setModelRot( modelRot )
72.                 < newState;
73.         }
74.     };
75. }
```

Listing 5.4 Signal function of ball 2 of 2

The trick to reading the simplified (incomplete) code snippet of ball signal function and its signal composition through combinatos (in form of overloaded operator "<", which is meant to resemble left arrow, to indicate that its right operand is being passed into its left operand) and other signal functions is to read them from right to left. This function can be seen in Listing 5.3 and **Error! Reference source not found.** aboveabove. For example, read from right to left, the code on line 15 takes input signal, applies it to the lifted function Listing 5.2 Getter of game input shown in Listing 5.2 above, which is a pure function.

Lines 22 through 26 in Listing 5.3, again read from right to left, take game input signal, which is previously obtained on line 15, Line 25 calls lifted version of the Getter of input state for "`Key::W`". Line 24 is a signal function that applies Boolean logical "AND" operator to two signals, one that is `onTheGround` that contains function from time to a `bool` value that is `true` if the ball is on the ground, and the other that is previously described Boolean signal of whether Key W is pressed. On line 23 the lifted function `ifElse` is a function that returns its first argument if the signal passed to evaluates to `true`, or the second argument if the signal is `false` (this higher-order function is shown in Listing 5.5 below). In this case, if the applied signal is `true` then the value of the resulting signal `accZForward` will be constant `acceleration`, otherwise if it is `false` then `accZForwrd` will be zero. This whole expression can be expressed in other words as: "If game input's key W is pressed and the ball is on the ground, acceleration on Z axis is equal to `acceleration` constant, zero otherwise.

```
1. template<typename A>
2. std::function<A( const bool )> ifElse( const A& a, const A&b )
3. {
4.     return [a, b]( const bool cond )
5.     {
6.         return cond ? a : b;
7.     };
8. }
```

Listing 5.5 Higher-order function ifElse

5.1.3 Implementation of Imperative Programming Version

A sample of actor definition in IP version, that is using actor component based system, can be seen in Listing 5.6 belowListing 5.1. First, it defines Ball actor made up of a custom derived class `BallTransformComponent` (lines 1-7) and `ModelComponent` (line 9-19). After that, it then defines Camera actor, which consists of `TransformComponent` (lines 20-26) and `CameraComponent` (lines 28-31), and adds it to the Ball's children (line 33).

```
1. BallTransformComponent ballTransform{ // transform
2.     { 0.0f, 0.45f, 0.0f }, // pos
3.     { 0.0f, 0.0f, 0.0f }, // vel
4.     { 1.0f, 1.0f, 1.0f }, // scl
5.     FQuat::identity, // rot
6.     FQuat::identity // modelRot
7. };
8. Actor ball( &ballTransform );
9. auto ballModel = ModelComponent( // model
10.     "basketball.fbx", // filename
11.     0.02f, // scale
12.     MaterialDef{ // material
13.     "basketball-diffuse.jpg", // diffuseTextureFilename
14.     "", // specularTextureFilename
15.     "basketball-bump.jpg", // bumpTextureFilename
16.     "", // parallaxTextureFilename
17.     "" // evnMapTextureFilename
18. } );
19. ball.addComponent( &ballModel );
20. TransformComponent cameraTransform{ // transform
21.     { 0.0f, 2.0f, -7.0f }, // pos
22.     { 0.0f, 0.0f, 0.0f }, // vel
23.     { 1.0f, 1.0f, 1.0f }, // scl
24.     FQuat::identity, // rot
25.     FQuat::identity // modelRot
26. };
27. Actor camera( &cameraTransform );
28. auto cameraComp = CameraComponent( { // camera
29.     0.001f, // nearClipDist
30.     1000.0f, // farClipDist
31. } );
32. camera.addComponent( &cameraComp );
33. ball.addChild( std::move( camera ) );
```

Listing 5.6 Actor definition in IP version

Simplified (incomplete) implementation of the `BallTransformComponent` from Listing 5.6 that derives from `TransformComponent` class is shown in next Listing 5.7. On lines 3 through 43 is overridden implementation of virtual `vUpdate` method, that defines the Ball's behaviour.

```
1. struct BallTransformComponent : public TransformComponent
2. {
3.     virtual void vUpdate( const float deltaMs,
4.         const GameInput& input ) override
5.     {
6.         static const float acceleration = 0.001f;
7.         static const float jumpSpeed = 0.01f;
8.         static const float rotSpeed = 0.003f;
9.         static const float minPosX = 0.45f;
10.        static const float gravity = -0.00000981f;
11.        // coefficient of restitution
12.        static const float cor = -0.5f;
13.        FVec3 acc = FVec3::zero;
14.        float velYUp = 0.0f;
15.        if ( _pos.y == minPosX )
16.        {
17.            if ( input[Key::W] )
18.            {
19.                acc.z += acceleration;
20.            }
21.            if ( input[Key::S] )
22.            {
23.                acc.z += -acceleration;
24.            }
25.            if ( input[Key::Space] )
26.            {
27.                velYUp = jumpSpeed;
28.            }
29.        }
30.        if ( input[Key::A] )
31.        {
32.            angVel += -rotSpeed;
33.        }
34.        if ( input[Key::D] )
35.        {
36.            angVel += rotSpeed;
37.        }
38.        _rot = _rot *
39.            eulerRadToQuat( FVec3{ 0.0f, angVel * deltaMs, 0.0f } );
40.        _modelRot = _modelRot *
41.            eulerRadToQuat( FVec3{ intVel.z, 0.0f, -intVel.x } *
42.            0.5f );
43.    }
44.};
```

Listing 5.7 Custom derived class in IP version

The logic implemented in Listing 5.7 above is a typical imperative implementation, where the logic is implemented as steps of statements or commands that the program execution follows and performs to mutate its state as opposed to FRP implementation

of signal functions, as a composition of expressions. This is where the two versions of the engine differ the most.

5.1.4 Application Profiling

The specifications of the machine used for profiling, optimisation and the other compiler settings can be found in Appendix B. The data measured during profiling can be found in form of graphs and tables in Appendix C.

Figure B.1 shows results of CPU profiling of FP version. It shows that the functions that take largest part of code execution time are lambda and closure functions such as signal function that seen in Listing 5.8. This happens most likely because the closures captures signal functions `fst` and `snd` by value, which creates new copies of them every time this function is called. On the other hand, the CPU profiling data of IP version can be found in Figure B.2. It shows that this version spends most of its execution time calling OS kernel functions, which means that the actual application code runs very efficiently. This is also backed-up by numerical comparison of CPU profiling data from FP and IP version in Figure B.5, categorised by the least efficient function calls, and Figure B.6 categorised by the modules of the least efficient function calls.

```
1.      template<typename A, typename B, typename C>
2.      SF<A, C> compose( const SF<A, B>& fst, const SF<B, C>& snd )
3.      {
4.          return SF < A, C > {
5.              [fst, snd]( const S<A>& a ) -> S < C >
6.              {
7.                  return snd.f( fst.f( a ) );
8.              }
9.          };
10.     }
```

Listing 5.8 One of the bottleneck signal functions in FP version

The data measured during memory profiling for FP and IP version are shown in Figure B.3 and Figure B.4 respectively. It shows that memory usage is lower in the FP version that is around 13.5 MB on average, while the IP version was using just under 15.0 MB on average.

5.1.5 Source Code Metrics

Number of source code lines of code for both version of the project show that the Functional Programming version was more verbose than the Imperative one with about 22% increase in the FP version over the IP.

Functional programming version: 3772 lines

Imperative programming version: 3115 lines

5.1.6 Evaluation Summary

The evaluation concludes that functional programming version of game engine written in C++ increases verbosity of the language. The measured data from profiling shown that adaption of functional programming concepts in C++ has negative effect on code

efficiency caused by inefficient capture method of closures that have been used throughout the FP version. It is possible that these issues could be resolved by planned features in upcoming C++ standards, such as move semantics for capture list of closures.

6 Discussion and Conclusions

To conclude the project a summary of its aims and findings is presented in the following sections. The questions it was not possible to explore and answer in this project are indicated in last section 6.3 that opens a dialog aimed at further research.

6.1 Project resume

Game engine development currently faces many challenges as the complexity of them is steadily increasing, even more with emergence of devices with multi-core processors, which are now considered standard for the most of them. The efforts to solve the issues of concurrency and parallelism lead to code that can often be difficult to read, maintain, and is usually tailored for its specific application. This in turn leads to interdependencies that break the code modularity. Functional programming is known to have solved some of the issues that modern software development faces, however in games development the theory of functional programming benefits has not been fully turned into practice as of yet. Therefore, the aim of the project was to examine and answer the research question and to test the accuracy its related hypotheses recapitulated below.

Research Question: *Can functional programming concepts be used in traditionally imperative language C++ to better manage complexity and thus improve 3D computer game engines development?*

H1: Functional programming concepts of first-class and higher-order functions, currying, immutable data, purity, algebraic and recursive data types, recursion, lazy evaluation, lambda expressions and closures can be used in C++ for game engine development.

H2: Functional programming concepts can be used in C++ game engine development to increase productivity, promote code modularity, extensibility, testability, reliability, simplify concurrency, and rapid prototyping.

H3: Functional programming concepts used in C++ game engine development will have negative effect on code performance.

It aimed to do so by development of two versions of game engine, one written in traditional imperative programming style, and the other in functional programming style, both with equal capabilities. These game engines in form of static libraries were then used to develop a demonstration application with well-defined "game-like" behaviour. The process of development of both versions of application, and data measured from profiling of the applications serves a base for the conclusions in the following section, that aims to conclude on the goals set out by the project.

6.2 Conclusions

The project has proven that functional programming concepts can to an extent be used in C++, however the perceivable benefit is outweighed by the limitation of the C++ language and lack of VC++ compiler support for features from new standards.

6.2.1 First Hypothesis

To try to reason about the accuracy of the first hypothesis, the findings in its focus area are presented below, preceded by the body of the hypothesis.

H1: Functional programming concepts of first-class and higher-order functions, currying, immutable data, purity, algebraic and recursive data types, recursion, lazy

evaluation, lambda expressions and closures can be used in C++ for game engine development.

The functional programming concepts that were successfully implemented in this project are:

- First-class & Higher-order Functions
- Lambda Expressions & Closures
- Recursive Data Types

The concepts that were implemented to a limited extent are:

- Immutable Data
- Purity
- Algebraic Data Types
- Recursive Functions

And finally the only concept that was not implemented is:

- Lazy Evaluation

The features from C++11 made it possible to make full use of First-class and Higher-order functions in C++, even though their syntax is not as concise as for example in Haskell. C++11 also adopted support for Lambda Expressions and Closures. However, the issue with functions in C++, that has negative effect on code readability and complexity, is that they can be expressed in many different formats, most of which are not interchangeable with the other. Recursive Data Types can be in C++ expressed without any limitation, as has been demonstrated for example in the actor hierarchy.

Immutable Data can be used in C++, however this property cannot be enforced and therefore it mostly relies on programmer's perseverance, which also applies to the rule of function purity.

It has been demonstrated that it is possible to develop and use limited Algebraic Data Type system, such as the one that has been for the functional programming version of the game engine. However, the process to do so is quite error-prone and difficult to debug in C++, which could be elevated by the planned feature for C++ called "Concepts" as detailed in section 3.3.1.3, which also explores how some of the benefits that "Concepts" could bring can be achieved, but additionally notes that some of them are not supported by the compiler used in this project and therefore could not be used. There are no known proven implementations of some more advanced concepts from functional programming such as monads in C++, that are important for fully pure function programming that is possible in Haskell.

Tail-recursive functions can in most cases be safely used as their optimisation is supported in most C++ compilers. However, the engine's main loop can be written as a recursive function, because it is effectively infinite and thus cannot be unrolled. Unrolling is the optimisation technique that C++ compilers use on tail-recursive functions. In Haskell, even recursive functions that are not tail-recursive, can be optimised because of lazy evaluation (O'Sullivan, et al., 2008).

Lazy evaluation is technique that obstacles reasoning about program's efficiency (Mena, 2014; O'Sullivan, et al., 2008) and therefore a performance critical soft real-time application that game engines are cannot afford to implement it.

Most of the C++ compilers support tail-recursive function optimisation and therefore this concept can be used in C++ as well.

The second hypothesis was to an extent accurate. The least beneficial concept from functional programming was found to be lazy evaluation that causes difficulties in reasoning about efficiency of code.

6.2.2 Second Hypothesis

The many areas of the second hypothesis are repeated below followed by discussion on every one of them.

H2: Functional programming concepts can be used in C++ game engine development to increase productivity, promote code modularity, extensibility, testability, reliability, simplify concurrency, and rapid prototyping.

6.2.2.1 Productivity

Increase in productivity was not perceived, as the functional programming version was at times much more verbose than the imperative programming version. This was mostly due to limited C++ template metaprogramming template type deduction that is difficult to overcome. Also, the in relation to the Functional Reactive Programming technique used in the functional programming version, it was more difficult to compose signals and signal functions, because every operation that in C++ uses arithmetic operator (+, -, *, /, %, ++, --) and structure member variable access (e.g. the last part of the following C++ code in Listing 6.1 C++ member variable access) or a constructor has to be redefined as a function to be possible to "lift" it into a signal function, unlike in Haskell where all these operations are defined as normal functions by default. Because of this and the difficulty and verbosity of C++ template metaprogramming the productivity has been observed to decrease.

```
struct Foo{ int bar; }; Foo foo{ 1337 }; int foobar = foo.bar;
```

Listing 6.1 C++ member variable access

The fact that C++ allows multiple different ways to define a function is problematic, because it does not improve flexibility of the language, on the contrary, it just tries to make up for its limitations that leads to more complications, some of which seriously harm readability of the C++ code, in contrast Haskell which uses uniform syntax for function definition and for anonymous function definition.

6.2.2.2 Code Modularity & Extensibility

The way the code logic is expressed in the functional programming version through Functional Reactive Programming is very modular and simple to extend and reuse, apart from the fact that it results in more verbose code as concluded in previous section 6.2.2.1. Reuse is possible through definition of behaviours as compositions of signal functions which are easily reusable. However, the way these expressions are evaluated in C++ harms performance of the application as has been shown in section 5.1.4. This limits the use of FRP for game engine development, unless this issue can be resolved.

6.2.2.3 Testability & Reliability

Because the way the functional programming engine is being used and extended, which is through pure functions only, it is easier to test, as pure functions do not have any hidden interdependencies that make testing difficult and error-prone. They are also more reliable because it is easier to reason about pure functions than impure functions.

6.2.2.4 Simplified Concurrency

It was not possible to explore this area of game engine development through use of functional programming concepts, due to reduced productivity during the implementation of this version, caused by limitations in C++ and compiler support. It would however make for a very interesting topic for further research.

6.2.2.5 Rapid Prototyping

As a consequence of reduced productivity concluded in section 6.2.2.1, functional programming did not result in any improvement in efficiency of prototype development or expressivity of the used language syntax. On top of that there were issues with running and debugging the functional programming version of the game engine directly from Visual Studio, because the demonstration application would slow down to a crawl with apparent disruptive stutters. It is estimated that the issues were caused by C++ convoluted implementation of lambda and closure functions.

6.2.2.6 Summary

The only area in which there was a perceivable benefit was in the direction of code modularity and extensibility however the very important area of simplified concurrency has not been possible to explore in scope of this project. Therefore it is not possible to answer the question, whether the way the user of the engine extends its behaviour through pure functions, would guarantee thread-safe code. This opens room for further dialog on this question as that is the area in which functional programming has proved to excel in different software development fields explored in more detail in section 1.1.

6.2.3 Third Hypothesis

The last hypothesis is again stated below.

H3: Functional programming concepts used in C++ game engine development will have negative effect on code performance.

This hypothesis has proven to be true as has been shown in finding backed up by measured data from profiling execution discussed in section 5.1.4. It is fully possible that these issues could be resolved, pointing to another area for further research in functional programming for game engine development in C++.

It can be argued that functional programming concepts implemented in C++ cause decrease in code efficiency not because they cannot be implemented inefficiently, but because compiler developers focus on implementing C++ standards' features and optimisation techniques that are most advantageous for imperative programming style. Imperative programming style is much common and ingrained in C++ development, especially game engine development, and that reduces feasibility of adoption of functional programming concepts in C++.

6.2.4 Final Conclusion

Can functional programming concepts be used in traditionally imperative language C++ to better manage complexity and thus improve 3D computer game engines development?

To finally conclude the research question, which is worded above, the answer to it has unfortunately proved to be mostly negative. However, this verdict is not final as C++ updates in standards C++11, C+14, and C++17 promise to increase the possibility of effectively using functional programming in C++ and in overcoming the issues found in this approach. This could also be addressed by further research as pointed out in the previous section.

6.3 Further Work

It is hoped that this project provides a reliable base for further research and development. Previous section pointed to areas explored in the following sections.

6.3.1 Continue Development

The application of functional programming in C++ for game engine development could be continued or attempted again in future to solve the issues that were not addressed in this project. Based on the research into the changes and development in C++ standard (as of yet unimplemented in VC++) that has been done for this project, it has been envisaged that the language support for functional programming will most likely improve in the future. Details about the project's public repository can be found in Appendix D. It is also possible that a different approach might prove more efficient.

6.3.2 Research Concurrency and Parallelism

The developed game engines and their applications could be further developed or expanded upon to explore techniques for multi-core concurrency and parallelism, which is repeatedly discussed topic not only at games conferences, but also in other fields of software development. The promise of simplified concurrency and parallelism is one of the strongest arguments for application of functional programming in games development, where application efficiency is a very important aspect. Specifically, it would be interesting to see if the process of implementing concurrency has been simplified and thus became less error-prone.

6.3.3 Compare Alternative Compilers

This project used VC++ compiler for development, which is known to be lacking behind on implementation of new features from updated C++ standards when compared to other compilers such as LLVM Clang, GNU GCC, Inter C++ Compiler and others. For example, unlike VC++, GCC compiler already has support for annotating pure functions and "Concepts Lite", important features for functional programming that were discussed in this project.

6.3.4 Consider Different Programming Languages

Beside comparison on alternative C++ compilers, it would be interesting to consider different programming languages that allow efficient implementation in functional programming style such as the recently released Rust language (Poss, 2014).

References

Acton, M., 2014. *Data-Oriented Design and C++ (CppCon 2014 keynote presentation).* [Online] Available at: https://www.youtube.com/watch?v=rX0ItVEVjHc

Albrecht, T., 2009. *Pitfalls of Object Oriented Programming presented at Game Connect: Asia Pacific 2009,* London: Sony Computer Entertainment Europe, Research & Development Division.

Alexandrescu, A., 2010. *The D Programming Language.* 1st ed. s.l.:Addison-Wesley Professional.

Backfield, J., 2014. *Becoming Functional.* 1st ed. s.l.:O'Reilly.

Backus, J., 1978. Can programming be liberated from the von Neumann style? A functional style and its algebra of programs.. *Communications of the ACM,* August, 21(8), pp. 613-641.

Balaam, A., 2012. Tail Call Optimisation in C++. *Overload,* 109(June), pp. 10-13.

Bauer, A. & Pizka, M., 2004. Tackling C++ Tail Calls. *Dr. Dobb's Digest*, 1 February.

Bernardy, J. P. et al., 2008. A comparison of c++ concepts and haskell type classes. *WGP '08 Proceedings of the ACM SIGPLAN workshop on Generic programming,* pp. 37-48.

Callaghan, P., 2012. Thinking Functionally with Haskell. *PragPub*, August, p. 18.

Carmack, J., 2012. *In-depth: Functional programming in C++.* [Online] Available at: http://gamasutra.com/view/news/169296/Indepth_Functional_programming_in_C.php

Carmack, J., 2013. *Quakecon 2013 keynote.* [Online] Available at: https://www.youtube.com/watch?v=Z2QgiQh9Fco

Chakravarty, M., 2008. *GHC/Type families.* [Online] Available at: https://wiki.haskell.org/GHC/Indexed_types

Courtney, A., Nilsson, H. & Peterson, J., 2003. *The Yampa Arcade, Haskell '03 Proceedings of the 2003 ACM SIGPLAN workshop on Haskell.* New York, ACM, pp. 7-18.

Dawson, B., 2012. *Don't Store That in a Float - Random ASCII.* [Online] Available at: http://randomascii.wordpress.com/2012/02/13/dont-store-that-in-a-float/
[Accessed 25 11 2014].

Deitel, P. J., Deitel, H. M. & Deitel, A., 2013. *C++11 for Programmers.* 2nd ed. s.l.:Prentice Hall.

Eberly, D. H., 2007. *3D Game Engine Design: A Practical Approach to Real-Time Computer Graphics.* 2nd ed. San Francisco: Morgan Kaufmann Publishers Inc..

Fleury, N., 2014. *C++ in Huge AAA Games (CppCon 2014).* [Online] Available at: https://www.youtube.com/watch?v=qYN6eduU06s

Fleury, N., 2014. *C++ in Huge AAA Games (CppCon 2014).* [Online] Available at: https://www.youtube.com/watch?v=qYN6eduU06s

Ford, N., 2014. *Functional Thinking.* 1st ed. s.l.:O'Reilly.

Frankau, S., Spinellis, D., Nassuphis, N. & Burgard, C., 2009. Commercial Uses: Going functional on exotic trades. *Journal of Functional Programming,* 19(1), pp. 27-45.

Fulgham, B. & Gouy, I., 2014. *The Computer Language Benchmarks Game.* [Online] Available at: http://benchmarksgame.alioth.debian.org/

Gockel, T., n.d. *C++ Pirate Cove - Lambda vs Bind.* [Online] Available at: http://www.gockelhut.com/c++/articles/lambda_vs_bind

Gregor, D., 2011. *ConceptGCC: concept extensions for C++.* [Online] Available at: http://www.generic-programming.org/software/ConceptGCC/

Gregor, D. et al., 2006. Concepts: linguistic support for generic programming in C++. *OOPSLA '06: Proceedings of the 21st Annual ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages, and Applications,* pp. 291-310.

Haffmans, A., 2013. Living with Lambdas, Functional Programming in C++. *PragPub*, July, pp. 43-54.

Harbour, J. S., 2010. *Multi-Threaded Game Engine Design.* 1st ed. s.l.:Course Technology PTR.

Hartel, P. & Henk, M., 1999. *Functional C.* 1 ed. s.l.:Addison-Wesley.

Horton, I., 2014. *Beginning C++.* 1st ed. s.l.:Apress.

Hudak, P., 2000. *The Haskell School of Expression: Learning Functional Programming through Multimedia.* New York: Cambridge University Press.

Hudak, P., Hughes, J., Peyton Jones, S. & Wadler, P., 2007. *A history of Haskell: being lazy with class.* New York, ACM, pp. 12-1-12-55 .

Hudak, P. & Jones, M. P., 1994. *Haskell vs. Ada vs. C++ vs. awk vs.. . . An experiment in software prototyping productivity,* New Haven: Yale University - Department of Computer Science.

Hughes, J., 1990. Why Functional Programming Matters. In: *Research Topics in Functional Programming.* s.l.:Addison-Wesley, pp. 17-42.

Hutton, G., 2007. *Programming in Haskell.* s.l.:Cambridge University Press.

Josuttis, N. M., 2012. *The C++ Standard Library: A Tutorial and Reference.* 2nd ed. s.l.:Addison-Wesley Professional.

Josuttis, N. M., 2012. *The C++ Standard Library: A Tutorial and Reference, Second Edition.* 2nd ed. Boston: Addison-Wesley Professional.

Kalev, D., 2009. *C++ Reference Guide.* [Online] Available at: http://www.informit.com/guides/content.aspx?g=cplusplus&seqNum=441

Krishnaswami, N. R., Benton, N. & Hoffman, J., 2012. *Higher-order functional reactive programming in bounded space, POPL '12 Proceedings of the 39th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages.* New York, ACM.

Lipovača, M., 2011. *Learn You a Haskell for Great Good!.* s.l.:No Starch Press.

Ljumovic, M., 2014. *C++ Multithreading Cookbook.* 1st ed. s.l.:Packt Publishing.

Lloyd, J. W., 1994. Practical advantages of declarative programming. *Joint Conference on Declarative Programming,* Volume 94, p. 94.

Lowensohn, J., 2010. *How Epic fir the Enreal Engie into the iPhone.* [Online] Available at: www.cnet.com/uk/news/how-epic-fit-the-unreal-engine-into-the-iphone [Accessed 24 December 2014].

Marlow, S., 2013. *Parallel and Concurrent Programming in Haskell.* s.l.:O'Reilly.

Marlow, S., Brandy, L., Coens, J. & Purdy, J., 2014. *There is no fork: an abstraction for efficient, concurrent, and concise data access.* New York, ACM.

McNamara, B. & Smaragdakis, Y., 2000. *FC++: Functional Programming in C++.* Montreal, International Conference on Functional Programming (ICFP).

McShaffry, M. & Graham, D., 2012. *Game Coding Complete.* 4th ed. s.l.:Delmar Learning.

Meijer, E., 2014. The Curse of the Excluded Middle. *Queue*, 26 April, 12(4), p. 20.

Meisinger, G., 2011. *File:Yampa signal functions.png - HaskellWiki.* [Online] Available at: https://wiki.haskell.org/File:Yampa_signal_functions.png [Accessed 3 February 2015].

Mena, A. S., 2014. *Beginning Haskell.* s.l.:Apress.

Microsoft, 2015. *Support For C++11 Feature (Modern C++).* [Online] Available at: https://msdn.microsoft.com/en-us/library/hh567368.aspx [Accessed 20 January 2015].

Milewski, B., 2011. *Haskell – The Pseudocode Language for C++ Template Metaprogramming.* Aspen, BoostCon.

Milewski, B., 2014. *C++17: I See a Monad in Your Future!.* [Online] Available at: http://bartoszmilewski.com/2014/02/26/c17-i-see-a-monad-in-your-future
[Accessed 20 1 2015].

Okasaki, C., 1998. *Purely Functional Data Structures.* 1st ed. s.l.:Cambridge University Press.

O'Sullivan, B., Stewart, D. & Goerzen, J., 2008. *Real World Haskell.* s.l.:O'Reilly.

Peyton Jones, S., 2003. *Wearing the hair shirt. A retrospective on Haskell (invited talk).* Louisiana, Conference on Principles of Programming Languages (POPL'03).

Peyton Jones, S., 2010. *Tackling the Awkward Squad: monadic input/output, concurrency, exceptions, and foreign-language calls in Haskell,* Cambridge: Microsoft Research.

Polukhin, A., 2013. *Boost C++ Application Development Cookbook.* 1st ed. s.l.:Packt Publishing.

Pop, I., 2010. *Experience report: Haskell as a reagent: results and observations on the use of Haskell in a Python project.* New York, ACM.

Poss, R., 2014. *Rust for functional programmers.* [Online] Available at: http://science.raphael.poss.name/rust-for-functional-programmers.html [Accessed 15 March 2015].

Probst, M., 2001. *Proper Tail Recursion in C,* Vienna: Vienna University of Technology.

Radzivilovksy, P., Galka, Y. & Novgrodov, S., 2015. *UTF-8 Everywhere manifesto.* [Online] Available at: http://utf8everywhere.org [Accessed 30 January 2015].

Reddy, M., 2011. *API Design for C++.* 1st ed. San Francisco: Morgan Kaufmann Publishers Inc..

Sankel, D., 2010. *Modern Functional Programming in C++,* Spencerport: Sankel Software.

Schuster, W., 2012. *Why Are You Not Using Functional Languages?*. [Online] Available at: http://www.infoq.com/research/functional-language-obstacles/

Seibel, P., 2005. *Practical Common Lisp*. 1st ed. s.l.:Apress.

Sherrod, A., 2006. *Ultimate 3D Game Engine Design & Architecture*. Rockland, MA, USA: Charles River Media, Inc..

Sherrod, A. & Jones, W., 2011. *Beginning DirectX 11 Game Programming*. 1st ed. Boston, MA, USA: Course Technology Press.

Smith, T., 2006. Recursive Descent, Tail Recursion, & the Dreaded Double Divide. *Dr. Dobb's Journal,* 31(1), pp. 36-39.

Solodkyy, Y., Reis, G. D. & Stroustrup, B., 2014. Open pattern matching for C++. *ACM SIGPLAN Notices - GPCE '13,* 45(3), pp. 33-42.

Stroustrup, B., 1993. *A history of C++: 1979–1991*. New York, ACM, pp. 271-297.

Stroustrup, B., 2013. *The C++ Programming Language*. 4th ed. s.l.:Addison-Wesley Professional.

Stroustrup, B., Dos Reis, G. & Sutton, A., 2013. *Concepts Lite: Constraining Templates with,* Texas: Texas A&M University, Department of Computer Science and Engineering.

Sutter, H., 2014. *Sutter's Mill - GotW #96: Oversharing*. [Online] Available at: http://herbsutter.com/2014/01/14/gotw-96-oversharing/

Sweeney, T., 2006. *The Next Mainstream Programming Language: A Game Developer's Perspective*. New York, ACM.

Trapni, 2013. *Performance of std::function compared to raw function pointer and void* this?*. [Online]
Available at: http://stackoverflow.com/a/19371276/3210255

Tsai, B.-Y., Stobart, S., Parrington, N. & Thompson, B., 1997. Iterative design and testing within the software development life cycle. *Software Quality Journal,* 6(4), pp. 295-310.

Tulip, J., Bekkema, J. & Nesbitt, K., 2006. Multi-threaded game engine design. *IE '06 Proceedings of the 3rd Australasian conference on Interactive entertainment,* pp. 9-14.

Veldhuizen, T. L., 1995. Using C++ template metaprograms. *C++ Report,* May, 7(4), pp. 36-43.

Wakely, J., 2013. *Stack Overflow - C ++ Boost Bind Performance*. [Online] Available at: http://stackoverflow.com/a/14310265

Walbourn, C., 2013. *Where is the DirectX SDK (2013 Edition)?*. [Online] Available at: http://blogs.msdn.com/b/chuckw/archive/2013/07/01/where-is-the-directx-sdk-2013-edition.aspx
[Accessed 12 December 2015].

Williams, A., 2012. *C++ Concurrency in Action: Practical Multithreading*. 1st ed. s.l.:Manning Publications.

Wyatt, P., 2012. *Avoiding game crashes related to linked lists*. [Online] Available at: http://www.codeofhonor.com/blog/avoiding-game-crashes-related-to-linked-lists

Bibliography

Abrahams, D. & Gurtovoy, A., 2004. *C++ Template Metaprogramming: Concepts, Tools, and Techniques from Boost and Beyond.* Boston: Addison-Wesley Professional.

Acton, M., 2014. *Data-Oriented Design and C++ (CppCon 2014 keynote presentation).* [Online] Available at: https://www.youtube.com/watch?v=rX0ItVEVjHc

Albrecht, T., 2009. *Pitfalls of Object Oriented Programming presented at Game Connect: Asia Pacific 2009,* London: Sony Computer Entertainment Europe, Research & Development Division.

Alexandrescu, A., 2010. *The D Programming Language.* 1st ed. s.l.:Addison-Wesley Professional.

Backfield, J., 2014. *Becoming Functional.* 1st ed. s.l.:O'Reilly.

Backus, J., 1978. Can programming be liberated from the von Neumann style? A functional style and its algebra of programs.. *Communications of the ACM,* August, 21(8), pp. 613-641.

Balaam, A., 2012. Tail Call Optimisation in C++. *Overload,* 109(June), pp. 10-13.

Bauer, A. & Pizka, M., 2004. Tackling C++ Tail Calls. *Dr. Dobb's Digest*, 1 February.

Bernardy, J. P. et al., 2008. A comparison of c++ concepts and haskell type classes. *WGP '08 Proceedings of the ACM SIGPLAN workshop on Generic programming,* pp. 37-48.

Callaghan, P., 2012. Thinking Functionally with Haskell. *PragPub*, August, p. 18.

Carmack, J., 2012. *In-depth: Functional programming in C++.* [Online] Available at: http://gamasutra.com/view/news/169296/Indepth_Functional_programming_in_C.php

Carmack, J., 2013. *Quakecon 2013 keynote.* [Online] Available at: https://www.youtube.com/watch?v=Z2QgiQh9Fco

Chakravarty, M., 2008. *GHC/Type families.* [Online] Available at: https://wiki.haskell.org/GHC/Indexed_types

Chakravarty, M., Keller, G. & Jones, S. P., 2005. *Associated Type Synonyms.* New York, ACM Press.

Chakravarty, M., Keller, G., Jones, S. P. & Marlow, S., 2005. *Associated types with class.* New York, ACM Press.

Courtney, A., Nilsson, H. & Peterson, J., 2003. *The Yampa Arcade, Haskell '03 Proceedings of the 2003 ACM SIGPLAN workshop on Haskell.* New York, ACM, pp. 7-18.

Dawson, B., 2012. *Don't Store That in a Float - Random ASCII.* [Online] Available at: http://randomascii.wordpress.com/2012/02/13/dont-store-that-in-a-float/
[Accessed 25 11 2014].

Deitel, P. J., Deitel, H. M. & Deitel, A., 2013. *C++11 for Programmers.* 2nd ed. s.l.:Prentice Hall.

Eberly, D. H., 2007. *3D Game Engine Design: A Practical Approach to Real-Time Computer Graphics.* 2nd ed. San Francisco: Morgan Kaufmann Publishers Inc..

Fleury, N., 2014. *C++ in Huge AAA Games (CppCon 2014).* [Online] Available at: https://www.youtube.com/watch?v=qYN6eduU06s

Fleury, N., 2014. *C++ in Huge AAA Games (CppCon 2014).* [Online] Available at: https://www.youtube.com/watch?v=qYN6eduU06s

Ford, N., 2014. *Functional Thinking.* 1st ed. s.l.:O'Reilly.

Frankau, S., Spinellis, D., Nassuphis, N. & Burgard, C., 2009. Commercial Uses: Going functional on exotic trades. *Journal of Functional Programming,* 19(1), pp. 27-45.

Fulgham, B. & Gouy, I., 2014. *The Computer Language Benchmarks Game.* [Online] Available at: http://benchmarksgame.alioth.debian.org/

Gockel, T., n.d. *C++ Pirate Cove - Lambda vs Bind.* [Online] Available at: http://www.gockelhut.com/c++/articles/lambda_vs_bind

Gregor, D., 2011. *ConceptGCC: concept extensions for C++.* [Online] Available at: http://www.generic-programming.org/software/ConceptGCC/

Gregor, D. et al., 2006. Concepts: linguistic support for generic programming in C++. *OOPSLA '06: Proceedings of the 21st Annual ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages, and Applications,* pp. 291-310.

Haffmans, A., 2013. Living with Lambdas, Functional Programming in C++. *PragPub*, July, pp. 43-54.

Harbour, J. S., 2010. *Multi-Threaded Game Engine Design.* 1st ed. s.l.:Course Technology PTR.

Hartel, P. & Henk, M., 1999. *Functional C.* 1 ed. s.l.:Addison-Wesley.

Horton, I., 2014. *Beginning C++.* 1st ed. s.l.:Apress.

Hudak, P., 2000. *The Haskell School of Expression: Learning Functional Programming through Multimedia.* New York: Cambridge University Press.

Hudak, P., Hughes, J., Peyton Jones, S. & Wadler, P., 2007. *A history of Haskell: being lazy with class.* New York, ACM, pp. 12-1-12-55 .

Hudak, P. & Jones, M. P., 1994. *Haskell vs. Ada vs. C++ vs. awk vs.. . . An experiment in software prototyping productivity,* New Haven: Yale University - Department of Computer Science.

Hughes, J., 1990. Why Functional Programming Matters. In: *Research Topics in Functional Programming.* s.l.:Addison-Wesley, pp. 17-42.

Hutton, G., 2007. *Programming in Haskell.* s.l.:Cambridge University Press.

Josuttis, N. M., 2012. *The C++ Standard Library: A Tutorial and Reference.* 2nd ed. s.l.:Addison-Wesley Professional.

Josuttis, N. M., 2012. *The C++ Standard Library: A Tutorial and Reference, Second Edition.* 2nd ed. Boston: Addison-Wesley Professional.

Kalev, D., 2009. *C++ Reference Guide.* [Online] Available at: http://www.informit.com/guides/content.aspx?g=cplusplus&seqNum=441

Krishnaswami, N. R., Benton, N. & Hoffman, J., 2012. *Higher-order functional reactive programming in bounded space, POPL '12 Proceedings of the 39th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages.* New York, ACM.

Lipovača, M., 2011. *Learn You a Haskell for Great Good!.* s.l.:No Starch Press.

Ljumovic, M., 2014. *C++ Multithreading Cookbook.* 1st ed. s.l.:Packt Publishing.

Lloyd, J. W., 1994. Practical advantages of declarative programming. *Joint Conference on Declarative Programming,* Volume 94, p. 94.

Lowensohn, J., 2010. *How Epic fit the Enreal Engie into the iPhone.* [Online] Available at: www.cnet.com/uk/news/how-epic-fit-the-unreal-engine-into-the-iphone [Accessed 24 December 2014].

Marlow, S., 2013. *Parallel and Concurrent Programming in Haskell.* s.l.:O'Reilly.

Marlow, S., Brandy, L., Coens, J. & Purdy, J., 2014. *There is no fork: an abstraction for efficient, concurrent, and concise data access.* New York, ACM.

McNamara, B. & Smaragdakis, Y., 2000. *FC++: Functional Programming in C++.* Montreal, International Conference on Functional Programming (ICFP).

McShaffry, M. & Graham, D., 2012. *Game Coding Complete.* 4th ed. s.l.:Delmar Learning.

Meijer, E., 2014. The Curse of the Excluded Middle. *Queue*, 26 April, 12(4), p. 20.

Meisinger, G., 2011. *File:Yampa signal functions.png - HaskellWiki.* [Online] Available at: https://wiki.haskell.org/File:Yampa_signal_functions.png [Accessed 3 February 2015].

Mena, A. S., 2014. *Beginning Haskell.* s.l.:Apress.

Microsoft, 2015. *Support For C++11 Feature (Modern C++).* [Online] Available at: https://msdn.microsoft.com/en-us/library/hh567368.aspx [Accessed 20 January 2015].

Milewski, B., 2011. *Haskell – The Pseudocode Language for C++ Template Metaprogramming.* Aspen, BoostCon.

Milewski, B., 2014. *C++17: I See a Monad in Your Future!.* [Online] Available at: http://bartoszmilewski.com/2014/02/26/c17-i-see-a-monad-in-your-future
[Accessed 20 1 2015].

Okasaki, C., 1998. *Purely Functional Data Structures.* 1st ed. s.l.:Cambridge University Press.

O'Sullivan, B., Stewart, D. & Goerzen, J., 2008. *Real World Haskell.* s.l.:O'Reilly.

Peyton Jones, S., 2003. *Wearing the hair shirt. A retrospective on Haskell (invited talk).* Louisiana, Conference on Principles of Programming Languages (POPL'03).

Peyton Jones, S., 2010. *Tackling the Awkward Squad: monadic input/output, concurrency, exceptions, and foreign-language calls in Haskell,* Cambridge: Microsoft Research.

Polukhin, A., 2013. *Boost C++ Application Development Cookbook.* 1st ed. s.l.:Packt Publishing.

Pop, I., 2010. *Experience report: Haskell as a reagent: results and observations on the use of Haskell in a Python project.* New York, ACM.

Poss, R., 2014. *Rust for functional programmers.* [Online] Available at: http://science.raphael.poss.name/rust-for-functional-programmers.html [Accessed 15 March 2015].

Prata, S., 2011. *C++ Primer Plus.* 6th ed. s.l.:Addison Wesley.

Probst, M., 2001. *Proper Tail Recursion in C,* Vienna: Vienna University of Technology.

Radzivilovksy, P., Galka, Y. & Novgrodov, S., 2015. *UTF-8 Everywhere manifesto.* [Online] Available at: http://utf8everywhere.org [Accessed 30 January 2015].

Reddy, M., 2011. *API Design for C++.* 1st ed. San Francisco: Morgan Kaufmann Publishers Inc..

Sankel, D., 2010. *Modern Functional Programming in C++,* Spencerport: Sankel Software.

Schuster, W., 2012. *Why Are You Not Using Functional Languages?.* [Online] Available at: http://www.infoq.com/research/functional-language-obstacles/

Seibel, P., 2005. *Practical Common Lisp.* 1st ed. s.l.:Apress.

Sherrod, A., 2006. *Ultimate 3D Game Engine Design & Architecture.* Rockland, MA, USA: Charles River Media, Inc..

Sherrod, A. & Jones, W., 2011. *Beginning DirectX 11 Game Programming.* 1st ed. Boston, MA, USA: Course Technology Press.

Smith, T., 2006. Recursive Descent, Tail Recursion, & the Dreaded Double Divide. *Dr. Dobb's Journal,* 31(1), pp. 36-39.

Solodkyy, Y., Reis, G. D. & Stroustrup, B., 2014. Open pattern matching for C++. *ACM SIGPLAN Notices - GPCE '13,* 45(3), pp. 33-42.

Stroustrup, B., 1993. *A history of C++: 1979–1991.* New York, ACM, pp. 271-297.

Stroustrup, B., 2013. *The C++ Programming Language.* 4th ed. s.l.:Addison-Wesley Professional.

Stroustrup, B., Dos Reis, G. & Sutton, A., 2013. *Concepts Lite: Constraining Templates with,* Texas: Texas A&M University, Department of Computer Science and Engineering.

Sutter, H., 2014. *Sutter's Mill - GotW #96: Oversharing.* [Online] Available at: http://herbsutter.com/2014/01/14/gotw-96-oversharing/

Sweeney, T., 2006. *The Next Mainstream Programming Language: A Game Developer's Perspective.* New York, ACM.

Trapni, 2013. *Performance of std::function compared to raw function pointer and void* this?.* [Online]
Available at: http://stackoverflow.com/a/19371276/3210255

Tsai, B.-Y., Stobart, S., Parrington, N. & Thompson, B., 1997. Iterative design and testing within the software development life cycle. *Software Quality Journal,* 6(4), pp. 295-310.

Tulip, J., Bekkema, J. & Nesbitt, K., 2006. Multi-threaded game engine design. *IE '06 Proceedings of the 3rd Australasian conference on Interactive entertainment,* pp. 9-14.

Veldhuizen, T. L., 1995. Using C++ template metaprograms. *C++ Report,* May, 7(4), pp. 36-43.

Wakely, J., 2013. *Stack Overflow - C ++ Boost Bind Performance.* [Online] Available at: http://stackoverflow.com/a/14310265

Walbourn, C., 2013. *Where is the DirectX SDK (2013 Edition)?.* [Online] Available at: http://blogs.msdn.com/b/chuckw/archive/2013/07/01/where-is-the-

directx-sdk-2013-edition.aspx
[Accessed 12 December 2015].

Williams, A., 2012. *C++ Concurrency in Action: Practical Multithreading*. 1st ed. s.l.:Manning Publications.

Wyatt, P., 2012. *Avoiding game crashes related to linked lists*. [Online] Available at: http://www.codeofhonor.com/blog/avoiding-game-crashes-related-to-linked-lists

Appendix A: Resources

Basketball 3D Model and Texture: http://tf3dm.com/3d-model/official-nba-spalding-basketball-86751.html

Wooden Tiles Textures: http://www.tutorialsforblender3d.com/Textures/Wood-Boards-NormalMap/Boards_Normal_2.html

Appendix B: Profiling Machine and Compiler

The profiling data presented in this appendix can be found in the project directory inside "docs" folder. The specifications of the computer used for profiling are shown in Table B.1below and the compiler settings in Table B.2. These settings can have great effect on results of the profiling measurements. For profiling measurements tool used was Visual Studio 2013 built-in profiling tools.

Table B.1 Profiling System specifications

Processor	Intel® Core™ i7-2670QM CPU @ 2.20 GHz
Number of Cores	4
Installed memory (RAM)	4.00GB
System Type	Windows 8.1 Pro 64-bit, x64-based processor
Graphics Card	NVidia GeForce GT 540M

Table B.2 Compiler settings

Platform	Windows 32-bit
Optimisation	Maximize Speed (/O2)
Inline Function Expansion	Default
Enable Intrinsic Functions	Yes (/Oi)
Favour Size Or Speed	Favour fast code (/Ot)
Omit Frame Pointers	No (/Oy-)
Enable Fiber-Safe Optimisations	No
Whole Program Optimisation	Yes (/GL)
Enable C++ Exceptions	No
Enable Run-Time Type Information	Yes (/GR)

Appendix C: Profiling Data

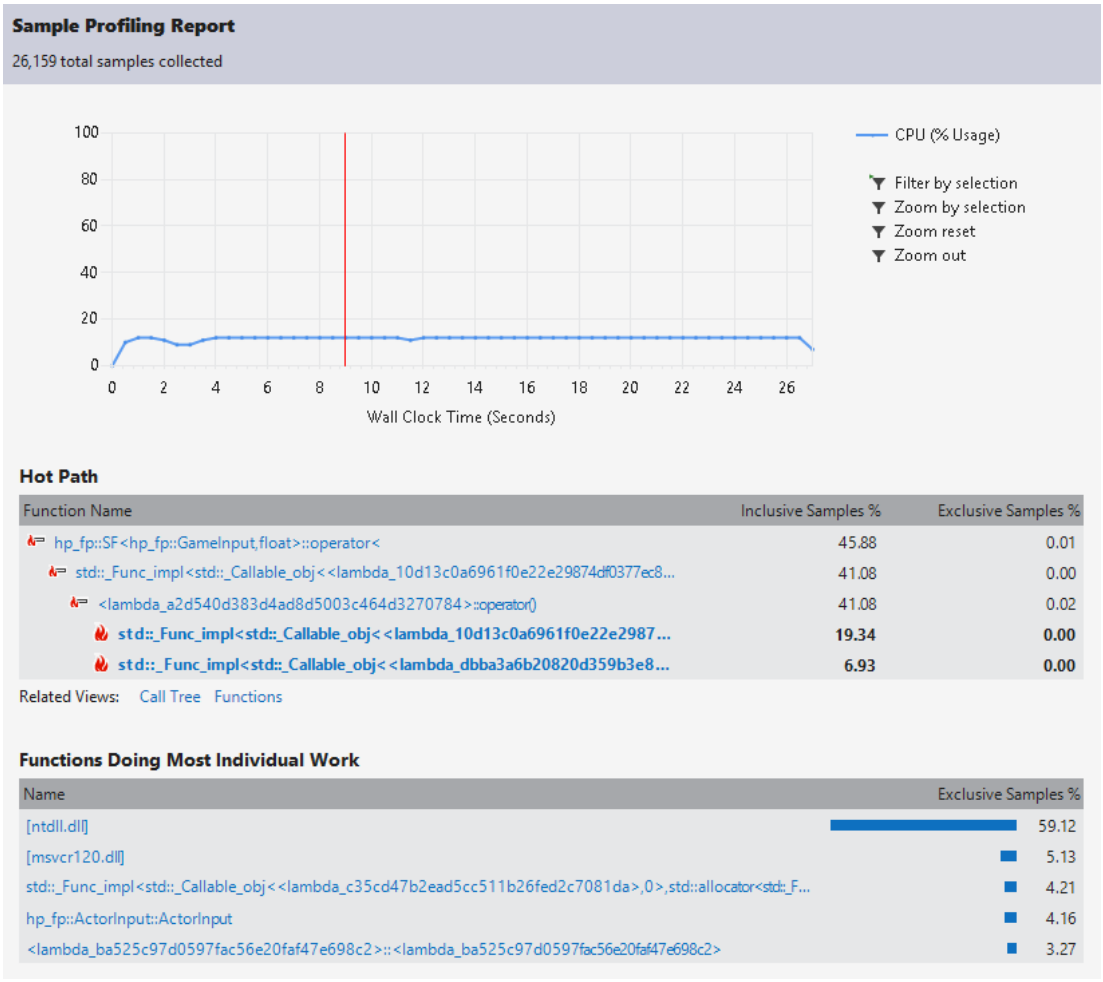


Figure B.1 CPU profiling of FP version

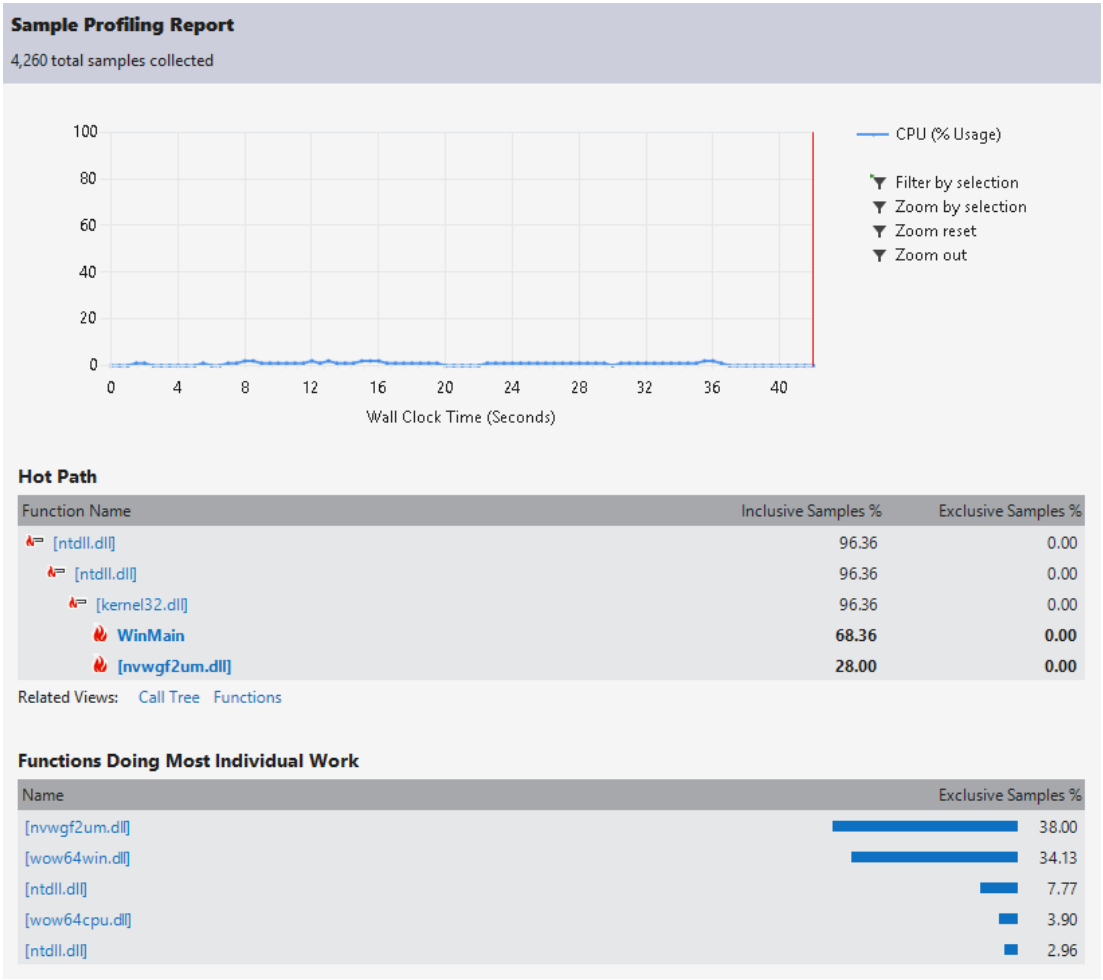


Figure B.2 CPU profiling of IP version

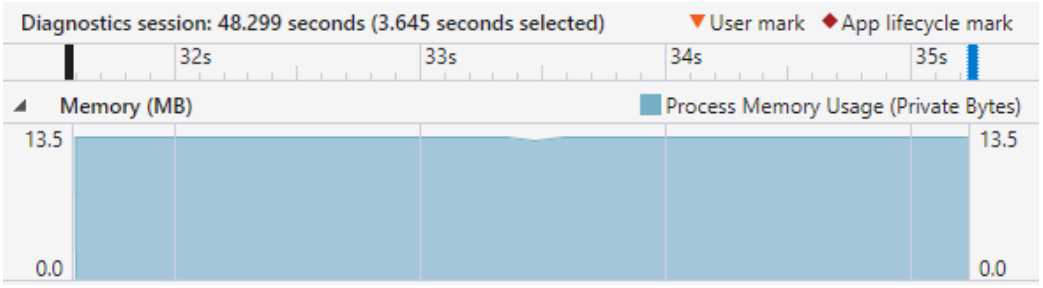


Figure B.3 Memory profiling of FP version

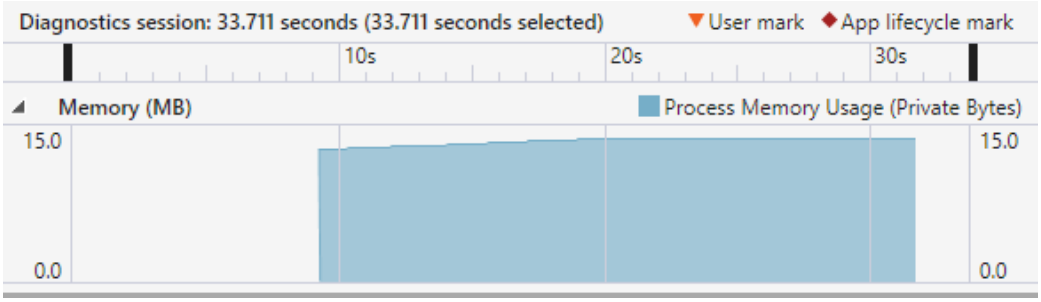
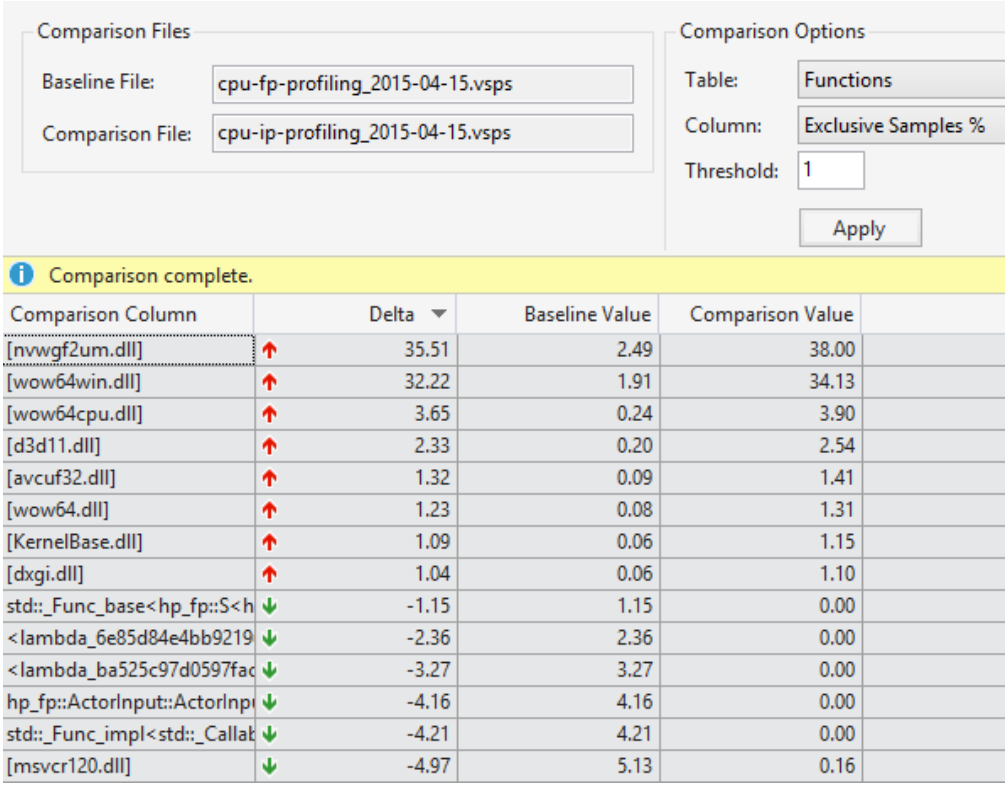
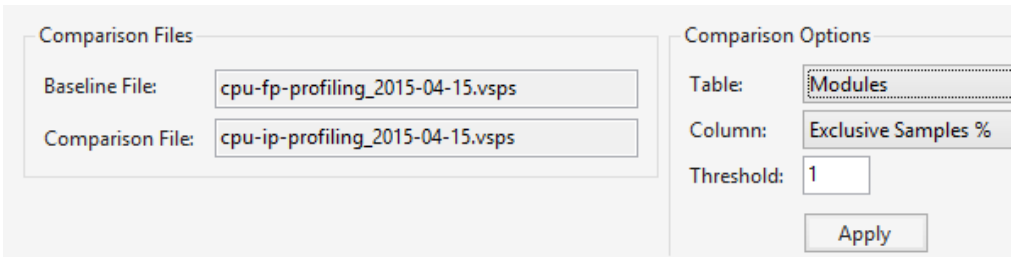


Figure B.4 Memory profiling of IP version



Comparison Column	Delta	Baseline Value	Comparison Value
[nvwgf2um.dll]	35.51	2.49	38.00
[wow64win.dll]	32.22	1.91	34.13
[wow64cpu.dll]	3.65	0.24	3.90
[d3d11.dll]	2.33	0.20	2.54
[avcuf32.dll]	1.32	0.09	1.41
[wow64.dll]	1.23	0.08	1.31
[KernelBase.dll]	1.09	0.06	1.15
[dxgi.dll]	1.04	0.06	1.10
std::_Func_base<hp_fp::S<h	-1.15	1.15	0.00
<lambda_6e85d84e4bb9219	-2.36	2.36	0.00
<lambda_ba525c97d0597fac	-3.27	3.27	0.00
hp_fp::ActorInput::ActorInp	-4.16	4.16	0.00
std::_Func_impl<std::_Callab	-4.21	4.21	0.00
[msvcr120.dll]	-4.97	5.13	0.16

Figure B.5 CPU profiling comparison of FP vs IP version categorised by the least efficient (often bottlneck) function calls



Comparison Column	Delta	Baseline Value	Comparison Value
[nvwgf2um.dll]	35.51	2.49	38.00
[wow64win.dll]	32.22	1.91	34.13
[wow64cpu.dll]	3.65	0.24	3.90
[d3d11.dll]	2.33	0.20	2.54
[avcuf32.dll]	1.32	0.09	1.41
[wow64.dll]	1.23	0.08	1.31
[KernelBase.dll]	1.09	0.06	1.15
[dxgi.dll]	1.04	0.06	1.10
std::_Func_base<hp_fp::S<h	-1.15	1.15	0.00
<lambda_6e85d84e4bb9219	-2.36	2.36	0.00
<lambda_ba525c97d0597fac	-3.27	3.27	0.00
hp_fp::ActorInput::ActorInp	-4.16	4.16	0.00
std::_Func_impl<std::_Callab	-4.21	4.21	0.00
[msvcr120.dll]	-4.97	5.13	0.16

Figure B.6 CPU profiling comparison of FP vs IP version categorised by the modules of the least efficient (often bottlneck) function calls

Appendix D: Open Project Repository

Project repository web URL: http://github.com/maseek/HP

Project git URL: https://github.com/maseek/HP.git

The directory hierarchy:

```
docs – comparison data from profiling measurements
functional – functional programming version
functional/3rdParty – third party libraries
functional/3rdParty/DirectXTex
functional/3rdParty/Effect11
functional/bin – compiled executables of the project's examples
functional/docs – project documentation files and profiling data
functional/include – project header files
functional/lib – compiled static library of the game engine
functional/msvc – Visual Studio solution and project files
functional/src – project source files
functional/temp – temporary files
imperative – imperative programming version
imperative/3rdParty – third party libraries such
imperative/3rdParty/DirectXTex
imperative/3rdParty/Effect11
imperative/bin – compiled executables of the project's examples
imperative/docs – project documentation files and profiling data
imperative/include – project header files
imperative/lib – compiled static library of the game engine
imperative/msvc – Visual Studio solution and project files
imperative/src – project source files
imperative/temp – temporary files
snippets – code snippets used throughout this report
tools – external tools used for development
```

Appendix E: Source Code of Functional Programming Version

```
#pragma once
#include <core/engine.hpp>
#include <adt/maybe.hpp>
#include <adt/sum.hpp>
#include <adt/frp/sfs.hpp>
#include <math/frustum.hpp>

#pragma once
namespace hp_fp
{
	template<typename A, typename B>
	struct Map
	{

	};
}

#pragma once
namespace hp_fp
{
	template<typename A>
	struct Maybe
	{
		template<typename B> friend Maybe<B> just( B&& b );
		template<typename B> friend Maybe<B> nothing( );
		template<typename B, typename C, typename D> friend auto ifThenElse( const
Maybe<B>& maybe, C ifJust, D ifNothing ) -> decltype( ifJust( *maybe._a ) );
		Maybe( const Maybe<A>& ) = delete;
		Maybe( Maybe&& m ) : _a( std::move( m._a ) )
		{
			m._a = nullptr;
		}
		Maybe<A> operator = ( const Maybe<A>& ) = delete;
		Maybe<A> operator = ( Maybe<A>&& m )
		{
			return Maybe<A>( std::move( m ) );
		}
		~Maybe( )
		{
			HP_DELETE( _a );
		}
	private:
		Maybe( A&& a ) : _a( HP_NEW A( std::move( a ) ) )
		{ }
		Maybe( ) : _a( nullptr )
		{ }
	private:
		A* _a;
	};
	template<typename A>
	Maybe<A> just( A&& a )
	{
		return Maybe<A>( std::move( a ) );
	}
	template<typename A>
	Maybe<A> nothing( )
	{
		return Maybe<A>( );
	}
	template<typename A, typename B, typename C>
	auto ifThenElse( const Maybe<A>& maybe, B ifJust, C ifNothing ) -> decltype( ifJust(
*maybe._a ) )
	{
		//static_assert( std::is_function<B>::value, "ifJust has to be a function."
);
		//static_assert( std::is_function<decltype( ifNothing )>::value, "ifNothing
has to be a function." );
		static_assert( std::is_same<decltype( ifJust( *maybe._a ) ), decltype(
ifNothing( ) )>::value, "ifJust and ifNothing functions' return types have to be the same."
);
```

```
                if ( maybe._a == nullptr )
                {
                        return ifNothing( );
                }
                return ifJust( *maybe._a );
        }
}

#pragma once
#include "../utils/typeId.hpp"
namespace hp_fp
{
        template<typename TypesHead, typename... TypesTail>
        struct SumHelper
        {
                inline static void destroy( const TypeId id, void * data )
                {
                        if ( id == typeId<TypesHead>( ) )
                        {
                                reinterpret_cast<TypesHead*>( data )->~TypesHead( );
                        }
                        else
                        {
                                SumHelper<TypesTail...>::destroy( id, data );
                        }
                }
                inline static void copy( const TypeId id, const void* oldData, void* newData
)
                {
                        if ( id == typeId<TypesHead>( ) )
                        {
                                new (newData) TypesHead( *reinterpret_cast<const
TypesHead*>( oldData ) );
                        }
                        else
                        {
                                SumHelper<TypesTail...>::copy( id, oldData, newData );
                        }
                }
                static const size_t val = sizeof( TypesHead ) > SumHelper<TypesTail...>::val
? sizeof( TypesHead ) : SumHelper<TypesTail...>::val;
        };
        template<typename Type>
        struct SumHelper < Type >
        {
                inline static void destroy( const TypeId id, void * data )
                {
                        reinterpret_cast<Type*>( data )->~Type( );
                }
                inline static void copy( const TypeId id, const void* oldData, void* newData
)
                {

                        const Type* t = reinterpret_cast<const Type*>( oldData );
                        new (newData) Type( *t );
                }
                static const size_t val = sizeof( Type );
        };
        template<typename... Types>
        struct Sum
        {
        public:
                template<typename Type>
                explicit Sum( Type&& value )
                {
                        new ( &_data ) Type( std::forward<Type>( value ) );
                        _typeId = typeId<Type>( );
                }
                template<typename Type>
                explicit Sum( const Type& value )
                {
                        new ( &_data ) Type( value );
                        _typeId = typeId<Type>( );
                }
```

```
                Sum( const Sum& s ) : _typeId( s._typeId )
                {
                        SumHelper<Types...>::copy( s._typeId, &s._data, &_data );
                }
                Sum( Sum&& s ) : _typeId( std::move( s._typeId ) ), _data( std::move( s._data
) )
                { }
                Sum operator = ( const Sum& s )
                {
                        return Sum( s );
                }
                Sum operator = ( Sum&& s )
                {
                        return Sum( std::move( s ) );
                }
                ~Sum( )
                {
                        SumHelper<Types...>::destroy( _typeId, &_data );
                }
                template<typename Type>
                bool is( ) const
                {
                        return ( _typeId == typeId<Type>( ) );
                }
                template<typename Type>
                Type& get( ) const
                {
                        return reinterpret_cast<Type&>( _data );
                }
        private:
                //const static size_t size = SumHelper<Types...>::val;
                TypeId _typeId;
        public:
                std::aligned_union<SumHelper<Types...>::val, Types...> _data;
        };
}

#pragma once
namespace hp_fp
{
        struct Unit
        { };
}

#pragma once
#include "../maybe.hpp"
namespace hp_fp
{
        template<typename A>
        using E = Maybe<A>;
        template<typename A>
        E<A> e( A&& a )
        {
                return E<A>( std::move( a ) );
        }
        template<typename A>
        E<A> noE( )
        {
                return E<A>( );
        }
}

#pragma once
namespace hp_fp
{
        template<typename A>
        struct S
        {
                S( std::function<A( const float )> f ) : f( f )
                { }
                S( const S& s ) : f( s.f )
                { }
                S( S&& s ) : f( std::move( s.f ) )
                { }
```

```
		S operator = ( const S& s )
		{
			return S{ s };
		}
		S operator = ( S&& s )
		{
			return S{ std::move( s ) };
		}
		std::function<A( const float )> f;
		A operator () ( const float deltaMs ) const
		{
			return f( deltaMs );
		}
		A operator < ( const float deltaMs ) const
		{
			return f( deltaMs );
		}
		A apply( const float deltaMs ) const
		{
			return f( deltaMs );
		}
	};
	/*}   }   }   }  }  }  } } } }}}} Functions {{{{ { { {  {  {  {   {   {   {*/

	template<typename A>
	S<A> constant( const A& a )
	{
		return S < A >
		{
			[a]( const float deltaMs ) -> A
			{
				return a;
			}
		};
	}
}

#pragma once
#include <functional>
#include "s.hpp"
namespace hp_fp
{
	template<typename A, typename B>
	struct SF
	{
		SF( std::function<S<B>( const S<A>& a )> f ) : f( f )
		{ }
		SF( const SF& sf ) : f( sf.f )
		{ }
		SF( SF&& sf ) : f( std::move( sf.f ) )
		{ }
		SF operator = ( const SF& sf )
		{
			return SF{ sf };
		}
		SF operator = ( SF&& sf )
		{
			return SF{ std::move( sf ) };
		}
		std::function<S<B>( const S<A>& a )> f;
		template<typename C>
		// compose two SF ( this >>> sf )
		SF<A, C> operator > ( SF<B, C>& sf ) const
		{
			return compose( *this, sf );
		}
		template<typename C>
		// compose two SF ( this <<< sf )
		SF<C, B> operator < ( SF<C, A>& sf ) const
		{
			return compose( sf, *this );
		}
		// apply signal to SF
		S<B> operator () ( const S<A>& a ) const
```

```
        {
                return f( a );
        }
        // apply constant value to SF
        S<B> operator () ( const A& a ) const
        {
                return f( constant( a ) );
        }
        // apply signal to SF
        S<B> operator < ( const S<A>& a ) const
        {
                return f( a );
        }
        // apply constant value to SF
        S<B> operator < ( const A& a ) const
        {
                return f( constant( a ) );
        }
};
template<typename B>
struct SF < void, B >
{
        std::function<S<B>( const S<void>& )> f;
        template<typename C>
        // compose two SF ( *this >>> sf )
        SF<void, C> operator > ( SF<B, C> sf )
        {
                return compose( *this, sf );
        }
};
/*}   }   }   }  }  }  } } } }}}} Functions {{{{ { { {  {  {  {   {   {   {*/

template<typename A, typename B>
SF<A, B> arr( B( *f )( const A& ) )
{
        return SF < A, B > {
                [f]( const S<A>& a ) -> S < B >
                {
                        return S < B > {
                                [f, a]( const float deltaMs )
                                {
                                        return f( std::forward<A>( a < deltaMs ) );
                                }
                        };
                }
        };
}
template<typename A, typename B>
SF<A, B> arrAlt( std::function<B( const A& )> f )
{
        return SF < A, B > {
                [f]( const S<A>& a ) -> S < B >
                {
                        return S < B > {
                                [f, a]( const float deltaMs )
                                {
                                        return f( std::forward<A>( a < deltaMs ) );
                                }
                        };
                }
        };
}
/*template<typename A, typename B>
SF<A, B> arr( std::function<B( const S<A>& )> f )
{
return SF < A, B > {
[f]( const S<A>& a )
{
return signal( f( a ), a.deltaMs );
}
};
}*/
template<typename A, typename B, typename C>
SF<A, C> compose( const SF<A, B>& fst, const SF<B, C>& snd )
```

```
        {
                return SF < A, C > {
                        [fst, snd]( const S<A>& a ) -> S < C >
                        {
                                return snd.f( fst.f( a ) );
                        }
                };
        }
}

#pragma once
#include <tuple>
#include "e.hpp"
#include "sf.hpp"
#include "../../math/vec3.hpp"
#include "../../math/quat.hpp"
namespace hp_fp
{
        template<typename A>
        SF<A, A> add( const S<A>& a )
        {
                return SF < A, A > {
                        [a]( const S<A>& b )
                        {
                                return S < A >
                                {
                                        [a, b]( const float deltaMs ) -> A
                                        {
                                                return a( deltaMs ) + b( deltaMs );
                                        }
                                };
                        }
                };
        }
        template<typename A>
        SF<A, A> add( const A& a )
        {
                return SF < A, A > {
                        [a]( const S<A>& b )
                        {
                                return S < A >
                                {
                                        [a, b]( const float deltaMs ) -> A
                                        {
                                                return a + b( deltaMs );
                                        }
                                };
                        }
                };
        }
        template<typename A>
        SF<A, A> sub( const S<A>& a )
        {
                return SF < A, A > {
                        [a]( const S<A>& b )
                        {
                                return S < A >
                                {
                                        [a, b]( const float deltaMs ) -> A
                                        {
                                                return a( deltaMs ) - b( deltaMs );
                                        }
                                };
                        }
                };
        }
        template<typename A>
        SF<A, A> sub( const A& a )
        {
                return SF < A, A > {
                        [a]( const S<A>& b )
                        {
                                return S < A >
                                {
```

```
                                [a, b]( const float deltaMs ) -> A
                                {
                                        return a - b( deltaMs );
                                }
                        }
                }
        };
}
template<typename A>
SF<A, A> mul( const S<A>& a )
{
        return SF < A, A > {
                [a]( const S<A>& b )
                {
                        return S < A >
                        {
                                [a, b]( const float deltaMs ) -> A
                                {
                                        return a( deltaMs ) * b( deltaMs );
                                }
                        };
                }
        };
}
template<typename A>
SF<A, A> mul( const A& a )
{
        return SF < A, A > {
                [a]( const S<A>& b )
                {
                        return S < A >
                        {
                                [a, b]( const float deltaMs ) -> A
                                {
                                        return a * b( deltaMs );
                                }
                        };
                }
        };
}
template<typename A, typename B>
SF<A, A> mul( const S<B>& b )
{
        return SF < A, A > {
                [b]( const S<A>& a )
                {
                        return S < A >
                        {
                                [a, b]( const float deltaMs ) -> A
                                {
                                        return a( deltaMs ) * b( deltaMs );
                                }
                        };
                }
        };
}
template<typename A, typename B>
SF<A, A> mul( const B& b )
{
        return SF < A, A > {
                [b]( const S<A>& a )
                {
                        return S < A >
                        {
                                [a, b]( const float deltaMs ) -> A
                                {
                                        return a( deltaMs ) * b;
                                }
                        };
                }
        };
}
template<typename A>
SF<A, A> div( const S<A>& a )
```

```
	{
		return SF < A, A > {
			[a]( const S<A>& b )
			{
				return S < A >
				{
					[a, b]( const float deltaMs ) -> A
					{
						return a( deltaMs ) / b( deltaMs );
					}
				};
			}
		};
	}
	template<typename A>
	SF<A, A> div( const A& a )
	{
		return SF < A, A > {
			[a]( const S<A>& b )
			{
				return S < A >
				{
					[a, b]( const float deltaMs ) -> A
					{
						return a / b( deltaMs );
					}
				};
			}
		};
	}
	template<typename A>
	SF<A, A> integral( )
	{
		return SF < A, A >
		{
			[]( const S<A>& a ) -> S < A >
			{
				return S < A >
				{
					[a]( const float deltaMs ) -> A
					{
						return a( deltaMs ) * deltaMs;
					}
				};
			}
		};
	}
	SF<FVec3, FVec3> rotate( const S<FQuat>& rot );
	SF<FVec3, FVec3> rotate( const FQuat& rot );

	template<typename A, typename B, typename C>
	SF<A, B> sw( const SF<A, std::tuple<B, E<C>>>& sf, std::function<SF<A, B>( C )> f )
	{
		return SF < A, B > {
			[sf, f]( const S<A>& a )
			{
				return S < B >
				{
					[sf, f, a]( const float deltaMs ) -> B
					{
						return ifThenElse( std::get<1>( sf < a <
deltaMs ),
							[f, a, deltaMs]( const C& c )
						{
							return f( c ) < a < deltaMs;
						},
							[sf, a, deltaMs]( )
						{
							return std::get<0>( sf < a < deltaMs
);
						} );
					}
				};
			}
```

```
                };
        }
}

#pragma once
#include "actor/actor.hpp"
#include "../window/gameInput.hpp"
#include "../window/window.hpp"
namespace hp_fp
{
        enum struct EngineState : UInt8
        {
                Initialized,
                Running,
                Terminated
        };
        // [const][cop-c][cop-a][mov-c][mov-a]
        // [  +  ][  0  ][  0  ][  +  ][  +  ]
        struct Engine
        {
                Engine( String&& name, EngineState&& state, GameInput&& gameInput )
                        : name( std::move( name ) ), state( std::move( state ) ),
                        gameInput( std::move( gameInput ) )
                { }
                Engine( const Engine& ) = delete;
                Engine( Engine&& e ) : name( std::move( e.name ) ), state( std::move( e.state
) ),
                        gameInput( std::move( e.gameInput ) )
                { }
                Engine operator = ( const Engine& ) = delete;
                Engine operator = ( Engine&& e )
                {
                        return Engine{ std::move( e ) };
                }
                const String name;
                EngineState state;
                GameInput gameInput;
        };
        /*}   }   }   }  }  }  } } } } }}}} Functions {{{{ { { {  {  {  {   {   {   {*/

        Engine init( String&& name );
        void run_IO( Engine& engine, std::vector<ActorDef>&& actorDefs,
                const WindowConfig& windowConfig = defaultWindowConfig_IO( ) );
        namespace
        {
                void renderActors_IO( Renderer& renderer, std::vector<Actor>& actors,
                        const GameInput& gameInput, const float deltaMs,
                        const Mat4x4& parentLocalTransform = Mat4x4::identity );
                std::vector<Actor> initActors_IO( Renderer& renderer, Resources& resources,
                        std::vector<ActorDef>&& actorsDef );
        }
}

#pragma once
#include <core/worldImm.hpp>
#include <window/gameInputMut.hpp>
namespace hp_fp
{

}#pragma once
#include <map>
#include <tuple>
#include "actor/actor.hpp"
#include "../graphics/model.hpp"
#include "../graphics/material.hpp"
namespace hp_fp
{
        // [const][cop-c][cop-a][mov-c][mov-a]
        // [  +  ][  0  ][  0  ][  0  ][  0  ]
        struct Resources
        {
                std::map<LoadedModelDef, Maybe<Model>> loadedModels;
                std::map<BuiltInModelDef, Maybe<Model>> builtInModels;
                std::map<MaterialDef, Maybe<Material>> materials;
```

```
                std::map<String, ID3D11ShaderResourceView*> textures;
        };
        struct ActorResources
        {
                Model& model;
                Material& material;
        };
        /*}   }   }   }  }  }  } } } }}}} Functions {{{{ { { {  {  {  {   {   {   {*/

        Maybe<Model>& getModel_IO( Renderer& renderer, Resources& resources,
                const LoadedModelDef& modelDef );
        Maybe<Model>& getModel_IO( Renderer& renderer, Resources& resources,
                const BuiltInModelDef& modelDef );
        Maybe<Material>& getMaterial_IO( Renderer& renderer, Resources& resources,
                const MaterialDef& materialDef );
        Maybe<ActorResources> getActorResources_IO( Renderer& renderer, Resources& resources,
                const ActorModelDef& actorModelDef );
        namespace
        {
                Maybe<ActorResources> getMaterialForModel_IO( Renderer& renderer, Resources&
resources,
                        Maybe<Model>& model, const MaterialDef& materialDef );
        }
}

#pragma once
namespace hp_fp
{
        extern const double TIME_ADDITION;
        // [const][cop-c][cop-a][mov-c][mov-a]
        // [  -  ][  0  ][  0  ][  +  ][  +  ]
        struct Timer
        {
                friend Timer initTimer_IO( );
                friend void updateTimer_IO( Timer& timer );
                Timer( const Timer& ) = delete;
                Timer operator = ( const Timer& ) = delete;
                Timer( Timer&& r ) : deltaMs( std::move( r.deltaMs ) ),
                        _timeMs( std::move( r._timeMs ) ), _lastTimeMs( std::move(
r._lastTimeMs ) )
                { }
                Timer operator = ( Timer&& r )
                {
                        return Timer{ r.deltaMs, r._timeMs, r._lastTimeMs };
                }
                double deltaMs;
                friend double timeMs( const Timer& timer );
        private:
                Timer( const double deltaMs, const double lastTimeMs, const double timeMs )
                        : deltaMs( deltaMs ), _lastTimeMs( lastTimeMs ), _timeMs( timeMs +
TIME_ADDITION )
                { }
                double _timeMs;
                double _lastTimeMs;
        };
        /*}   }   }   }  }  }  } } } }}}} Functions {{{{ { { {  {  {  {   {   {   {*/

        Timer initTimer_IO( );
        void updateTimer_IO( Timer& timer );
        double timeMs( const Timer& timer );
        namespace
        {
                double getTimeMs_IO( );
        }
}#pragma once
#include <functional>
#include <vector>
#include "../../adt/frp/sf.hpp"
#include "../../graphics/material.hpp"
#include "../../graphics/model.hpp"
#include "../../graphics/renderer.hpp"
#include "../../math/mat4x4.hpp"
#include "../../math/quat.hpp"
#include "../../window/gameInput.hpp"
```

```
namespace hp_fp
{
	struct Resources;
	struct ActorResources;
	struct ActorStartingState
	{
		FVec3 pos;
		FVec3 vel;
		FVec3 scl;
		FQuat rot;
		FQuat modelRot;
	};
	struct ActorState
	{
		FVec3 pos;
		FVec3 vel;
		FVec3 scl;
		FQuat rot;
		FQuat modelRot;
	};
	struct ActorInput
	{
		GameInput gameInput;
		ActorState state;
	};
	struct ActorOutput
	{
		ActorState state;
	};
	struct ActorModelDef
	{
		ModelDef model;
		MaterialDef material;
	};
	typedef std::function<void( Renderer&, const ActorState&, const Mat4x4& )>
CamRenderFn;
	struct ActorCameraDef;
	typedef CamRenderFn( *InitCamRenderFn )( const ActorCameraDef&, const WindowConfig&
);
	struct ActorCameraDef
	{
		float nearClipDist;
		float farClipDist;
		InitCamRenderFn render;
	};
	struct ActorTypeDef
	{
		template<typename A>
		bool is( ) const
		{
			return ( _typeId == typeId<A>( ) );
		}
	private:
		TypeId _typeId;
	public:
		union
		{
			ActorModelDef model;
			ActorCameraDef camera;
		};
		friend ActorTypeDef actorModelDef( ActorModelDef&& m );
		friend ActorTypeDef actorCameraDef( ActorCameraDef&& c );
	};
	struct ActorDef
	{
		ActorTypeDef type;
		ActorStartingState startingState;
		SF<ActorInput, ActorOutput> sf;
		std::vector<ActorDef> children;
	};
	struct Actor
	{
		ActorState state;
		SF<ActorInput, ActorOutput> sf;
```

```
                std::function<void( Renderer&, const ActorState&, const Mat4x4& )> render_IO;
                std::vector<Actor> children;
        };
        /*}   }   }   }  }  }  } } } }}}} Functions {{{{ { { {  {  {  {   {   {   {*/

        ActorTypeDef actorModelDef( ActorModelDef&& m );
        ActorTypeDef actorCameraDef( ActorCameraDef&& c );
        std::function<void( Renderer&, const ActorState&, const Mat4x4& )>
                initActorRenderFunction_IO( Renderer& renderer, Resources& resources,
                const ActorDef& actorDef );
        Mat4x4 trasformMatFromActorState( const ActorState& actorState );
        Mat4x4 modelTrasformMatFromActorState( const ActorState& actorState );
        namespace
        {
                std::function<void( Renderer&, const ActorState&, const Mat4x4& )>
renderActor_IO(
                        ActorResources& res );
        }
}

#pragma once
#include "../math/mat4x4.hpp"
namespace hp_fp
{
        struct Camera
        {
                Mat4x4 projection;
                Mat4x4 transform;
        };
        struct CameraBuffer
        {
        private:
                bool _first;
                Camera _cam[2];
        public:
                friend const Camera& getCamera( const CameraBuffer& cameraBuffer );
                friend void setCamera_IO( CameraBuffer& cameraBuffer, Camera&& camera );
                friend void swap_IO( CameraBuffer& cameraBuffer );
        };
        /*}   }   }   }  }  }  } } } }}}} Functions {{{{ { { {  {  {  {   {   {   {*/

        const Camera& getCamera( const CameraBuffer& cameraBuffer );
        void setCamera_IO( CameraBuffer& cameraBuffer, Camera&& camera );
        void swap_IO( CameraBuffer& cameraBuffer );
}

#pragma once
#include <d3d11_2.h>
#pragma comment(lib, "d3d11.lib")
#include <D3Dcompiler.h>
#pragma comment(lib, "d3dcompiler.lib")
#include <Effects11/inc/d3dx11effect.h>
#pragma comment(lib, "Effects11.lib")
#include "DirectXTex/DDSTextureLoader/DDSTextureLoader.h"
#include "DirectXTex/WICTextureLoader/WICTextureLoader.h"

#pragma once
#include "directx.hpp"
#include "renderer.hpp"
#include "../math/mat4x4.hpp"
#include "../math/color.hpp"
#include "../math/vec2.hpp"
namespace hp_fp
{
        struct MaterialDef
        {
                const char* diffuseTextureFilename;
                const char* specularTextureFilename;
                const char* bumpTextureFilename;
                const char* parallaxTextureFilename;
                const char* evnMapTextureFilename;
                FVec2 textureRepeat;
                bool operator == ( const MaterialDef& m ) const
                {
```

```
                        return diffuseTextureFilename == m.diffuseTextureFilename &&
                                specularTextureFilename == m.specularTextureFilename &&
                                bumpTextureFilename == m.bumpTextureFilename &&
                                parallaxTextureFilename == m.parallaxTextureFilename &&
                                evnMapTextureFilename == m.evnMapTextureFilename &&
                                textureRepeat == m.textureRepeat;
                }
                bool operator < ( const MaterialDef& m ) const
                {
                        if ( diffuseTextureFilename == m.diffuseTextureFilename )
                        {
                                if ( specularTextureFilename == m.specularTextureFilename )
                                {
                                        if ( bumpTextureFilename == m.bumpTextureFilename )
                                        {
                                                if ( parallaxTextureFilename ==
m.parallaxTextureFilename )
                                                {
                                                        if ( evnMapTextureFilename ==
m.evnMapTextureFilename )
                                                        {
                                                                return  textureRepeat <
m.textureRepeat;
                                                        }
                                                        return evnMapTextureFilename <
m.evnMapTextureFilename;
                                                }
                                                return parallaxTextureFilename <
m.parallaxTextureFilename;
                                        }
                                        return bumpTextureFilename < m.bumpTextureFilename;
                                }
                                return specularTextureFilename < m.specularTextureFilename;
                        }
                        return diffuseTextureFilename < m.diffuseTextureFilename;
                }
        };
        // [const][cop-c][cop-a][mov-c][mov-a]
        // [  +  ][  0  ][  0  ][  +  ][  +  ]
        struct Material
        {
                Material( const String& filename, const String& techniqueName,
                        const FVec2& textureRepeat, const Color& ambientMaterial,
                        const Color& diffuseMaterial, const Color& specularMaterial,
                        const float specularPower ) : filename( filename ), techniqueName(
techniqueName ),
                        effect( nullptr ), technique( nullptr ), pass( nullptr ),
inputLayout( nullptr ),
                        techniqueDesc( ), viewMatrixVariable( nullptr ),
projectionMatrixVariable( nullptr ),
                        worldMatrixVariable( nullptr ), diffuseTextureVariable( nullptr ),
                        specularTextureVariable( nullptr ), bumpTextureVariable( nullptr ),
                        parallaxTextureVariable( nullptr ), envMapVariable( nullptr ),
                        diffuseTexture( nullptr ), specularTexture( nullptr ), bumpTexture(
nullptr ),
                        parallaxTexture( nullptr ), envMapTexture( nullptr ), textureRepeat(
textureRepeat ),
                        useDiffuseTextureVariable( nullptr ), useSpecularTextureVariable(
nullptr ),
                        useBumpTextureVariable( nullptr ), useParallaxTextureVariable(
nullptr ),
                        ambientLightColourVariable( nullptr ), diffuseLightColourVariable(
nullptr ),
                        specularLightColourVariable( nullptr ), lightDirectionVariable(
nullptr ),
                        ambientMaterialVariable( nullptr ), diffuseMaterialVariable( nullptr
),
                        specularMaterialVariable( nullptr ), specularPowerVariable( nullptr
),
                        textureRepeatVariable( nullptr ), cameraPositionVariable( nullptr ),
                        ambientMaterial( ambientMaterial ), diffuseMaterial( diffuseMaterial
),
                        specularMaterial( specularMaterial ), specularPower( specularPower )
                {
```

```
                ZeroMemory( &techniqueDesc, sizeof( D3D10_TECHNIQUE_DESC ) );
        }
        Material( const Material& ) = delete;
        Material( Material&&  m ) : filename( std::move( m.filename ) ),
                techniqueName( std::move( m.techniqueName ) ), effect( std::move(
m.effect ) ),
                technique( std::move( m.technique ) ), pass( std::move( m.pass ) ),
                inputLayout( std::move( m.inputLayout ) ),
                techniqueDesc( std::move( m.techniqueDesc ) ),
                viewMatrixVariable( std::move( m.viewMatrixVariable ) ),
                projectionMatrixVariable( std::move( m.projectionMatrixVariable ) ),
                worldMatrixVariable( std::move( m.worldMatrixVariable ) ),
                diffuseTextureVariable( std::move( m.diffuseTextureVariable ) ),
                specularTextureVariable( std::move( m.specularTextureVariable ) ),
                bumpTextureVariable( std::move( m.bumpTextureVariable ) ),
                parallaxTextureVariable( std::move( m.parallaxTextureVariable ) ),
                envMapVariable( std::move( m.envMapVariable ) ),
                diffuseTexture( std::move( m.diffuseTexture ) ),
                specularTexture( std::move( m.specularTexture ) ),
                bumpTexture( std::move( m.bumpTexture ) ),
                parallaxTexture( std::move( m.parallaxTexture ) ),
                envMapTexture( std::move( m.envMapTexture ) ),
                textureRepeat( std::move( m.textureRepeat ) ),
                useDiffuseTextureVariable( std::move( m.useDiffuseTextureVariable )
),
                useSpecularTextureVariable( std::move( m.useSpecularTextureVariable
) ),
                useBumpTextureVariable( std::move( m.useBumpTextureVariable ) ),
                useParallaxTextureVariable( std::move( m.useParallaxTextureVariable
) ),
                ambientLightColourVariable( std::move( m.ambientLightColourVariable
) ),
                diffuseLightColourVariable( std::move( m.diffuseLightColourVariable
) ),
                specularLightColourVariable( std::move(
m.specularLightColourVariable ) ),
                lightDirectionVariable( std::move( m.lightDirectionVariable ) ),
                ambientMaterialVariable( std::move( m.ambientMaterialVariable ) ),
                diffuseMaterialVariable( std::move( m.diffuseMaterialVariable ) ),
                specularMaterialVariable( std::move( m.specularMaterialVariable ) ),
                specularPowerVariable( std::move( m.specularPowerVariable ) ),
                textureRepeatVariable( std::move( m.textureRepeatVariable ) ),
                cameraPositionVariable( std::move( m.cameraPositionVariable ) ),
                ambientMaterial( std::move( m.ambientMaterial ) ),
                diffuseMaterial( std::move( m.diffuseMaterial ) ),
                specularMaterial( std::move( m.specularMaterial ) ),
                specularPower( std::move( m.specularPower ) )
        {
                m.effect = nullptr;
                m.technique = nullptr;
                m.pass = nullptr;
                m.inputLayout = nullptr;
                m.viewMatrixVariable = nullptr;
                m.projectionMatrixVariable = nullptr;
                m.worldMatrixVariable = nullptr;
                m.diffuseTextureVariable = nullptr;
                m.specularTextureVariable = nullptr;
                m.bumpTextureVariable = nullptr;
                m.parallaxTextureVariable = nullptr;
                m.envMapVariable = nullptr;
                m.diffuseTexture = nullptr;
                m.specularTexture = nullptr;
                m.bumpTexture = nullptr;
                m.parallaxTexture = nullptr;
                m.envMapTexture = nullptr;
                m.useDiffuseTextureVariable = nullptr;
                m.useSpecularTextureVariable = nullptr;
                m.useBumpTextureVariable = nullptr;
                m.useParallaxTextureVariable = nullptr;
                m.ambientLightColourVariable = nullptr;
                m.diffuseLightColourVariable = nullptr;
                m.specularLightColourVariable = nullptr;
                m.lightDirectionVariable = nullptr;
                m.ambientMaterialVariable = nullptr;
```

```
                m.diffuseMaterialVariable = nullptr;
                m.specularMaterialVariable = nullptr;
                m.specularPowerVariable = nullptr;
                m.textureRepeatVariable = nullptr;
                m.cameraPositionVariable = nullptr;
        }
        Material operator = ( const Material& ) = delete;
        Material operator = ( Material&& m )
        {
                return Material( std::move( m ) );
        }
        ~Material( )
        {
                HP_RELEASE( envMapTexture );
                HP_RELEASE( parallaxTexture );
                HP_RELEASE( bumpTexture );
                HP_RELEASE( specularTexture );
                HP_RELEASE( diffuseTexture );
                HP_RELEASE( inputLayout );
                HP_RELEASE( effect );
        }
        String filename;
        String techniqueName;
        // effect variables
        ID3DX11Effect* effect;
        ID3DX11EffectTechnique* technique;
        ID3DX11EffectPass* pass;
        // input layout
        ID3D11InputLayout* inputLayout;
        // technique desc
        D3DX11_TECHNIQUE_DESC techniqueDesc;
        // effect variables(constants)
        ID3DX11EffectMatrixVariable* viewMatrixVariable;
        ID3DX11EffectMatrixVariable* projectionMatrixVariable;
        ID3DX11EffectMatrixVariable* worldMatrixVariable;
        // Textures
        ID3DX11EffectShaderResourceVariable* diffuseTextureVariable;
        ID3DX11EffectShaderResourceVariable* specularTextureVariable;
        ID3DX11EffectShaderResourceVariable* bumpTextureVariable;
        ID3DX11EffectShaderResourceVariable* parallaxTextureVariable;
        ID3DX11EffectShaderResourceVariable* envMapVariable;
        ID3D11ShaderResourceView* diffuseTexture;
        ID3D11ShaderResourceView* specularTexture;
        ID3D11ShaderResourceView* bumpTexture;
        ID3D11ShaderResourceView* parallaxTexture;
        ID3D11ShaderResourceView* envMapTexture;
        FVec2 textureRepeat;
        // Texture switches
        ID3DX11EffectScalarVariable* useDiffuseTextureVariable;
        ID3DX11EffectScalarVariable* useSpecularTextureVariable;
        ID3DX11EffectScalarVariable* useBumpTextureVariable;
        ID3DX11EffectScalarVariable* useParallaxTextureVariable;
        // Light
        ID3DX11EffectVectorVariable* ambientLightColourVariable;
        ID3DX11EffectVectorVariable* diffuseLightColourVariable;
        ID3DX11EffectVectorVariable* specularLightColourVariable;
        // Direction
        ID3DX11EffectVectorVariable* lightDirectionVariable;
        // Material
        ID3DX11EffectVectorVariable* ambientMaterialVariable;
        ID3DX11EffectVectorVariable* diffuseMaterialVariable;
        ID3DX11EffectVectorVariable* specularMaterialVariable;
        ID3DX11EffectScalarVariable* specularPowerVariable;
        ID3DX11EffectVectorVariable* textureRepeatVariable;
        // Camera
        ID3DX11EffectVectorVariable* cameraPositionVariable;
        // Material colours
        Color ambientMaterial;
        Color diffuseMaterial;
        Color specularMaterial;
        float specularPower;
};
/*}   }   }   }  }  }  } } } }}}} Functions {{{{ { { {  {  {  {   {   {   {*/
```

```
        Material defaultMat( );
        Maybe<Material> loadMaterial_IO( Renderer& renderer, const MaterialDef& materialDef
);
        bool loadTexture_IO( ID3D11ShaderResourceView** texture, Renderer& renderer,
                const String& filename );
        namespace
        {
                bool initMaterial_IO( Material& material, Renderer& renderer );
                bool loadShader_IO( Material& material, Renderer& renderer );
                bool loadAndCompile_IO( Material& material, Renderer& renderer, const String&
shaderModel,
                        ID3DBlob** buffer );
                bool createVertexLayout_IO( Material& material, Renderer& renderer );
        }
        void setProjection_IO( Material& material, const Mat4x4& mat );
        void setView_IO( Material& material, const Mat4x4& mat );
        void setWorld_IO( Material& material, const Mat4x4& mat );
        void setAbientLightColor_IO( Material& material, const Color& color );
        void setDiffuseLightColor_IO( Material& material, const Color& color );
        void setSpecularLightColor_IO( Material& material, const Color& color );
        void setLightDirection_IO( Material& material, const FVec3& dir );
        void setCameraPosition_IO( Material& material, const FVec3& dir );
        void setTextureRepeat_IO( Material& material, const FVec2& repeat );
        void setTextures_IO( Material& material );
        void setMaterials_IO( Material& material );
        void bindInputLayout_IO( Renderer& renderer, Material& material );
        UInt32 getPassCount( Material& material );
        void applyPass_IO( Renderer& renderer, Material& material, UInt32 i );
}

#pragma once
#include <vector>
#include "vertex.hpp"
#include "../adt/list.hpp"
#include "../adt/maybe.hpp"
#include "../adt/sum.hpp"
#include "../math/vec3.hpp"
#include "../utils/typeId.hpp"
namespace hp_fp
{
        struct Renderer;
        enum struct BuiltInModelType : UInt8
        {
                Cube, Sphere, Count
        };
        struct BuiltInModelDef
        {
                BuiltInModelType type;
                FVec3 dimensions;
                bool operator == ( const BuiltInModelDef& m ) const
                {
                        return type == m.type && dimensions == m.dimensions;
                }
                bool operator < ( const BuiltInModelDef& m ) const
                {
                        if ( type == m.type )
                        {
                                return  dimensions < m.dimensions;
                        }
                        return type < m.type;
                }
        };
        struct LoadedModelDef
        {
                const char* filename;
                float scale;
                bool operator == ( const LoadedModelDef& m ) const
                {
                        return filename == m.filename && scale == m.scale;
                }
                bool operator < ( const LoadedModelDef& m ) const
                {
                        if ( filename == m.filename )
                        {
```

```
                                return  scale < m.scale;
                        }
                        return filename < m.filename;
                }
        };
        //typedef Sum<BuiltInModelDef, LoadedModelDef> ModelDef;
        struct ModelDef
        {
                template<typename A>
                bool is( ) const
                {
                        return ( _typeId == typeId<A>( ) );
                }
        private:
                TypeId _typeId;
        public:
                union
                {
                        BuiltInModelDef builtIn;
                        LoadedModelDef loaded;
                };
                friend ModelDef builtInModelDef( BuiltInModelDef&& m );
                friend ModelDef loadedModelDef( LoadedModelDef&& m );
        };
        // [const][cop-c][cop-a][mov-c][mov-a]
        // [  +  ][  0  ][  0  ][  +  ][  +  ]
        struct Mesh
        {
                Mesh( ) : vertexBuffer( nullptr ), indexBuffer( nullptr )
                { }
                Mesh( const Mesh& ) = delete;
                Mesh( Mesh&&  m ) : vertices( std::move( m.vertices ) ), indices( std::move(
m.indices ) ),
                        vertexBuffer( std::move( m.vertexBuffer ) ), indexBuffer( std::move(
m.indexBuffer ) )
                { }
                Mesh operator = ( const Mesh& ) = delete;
                Mesh operator = ( Mesh&& m )
                {
                        return Mesh( std::move( m ) );
                }
                ~Mesh( )
                {
                        HP_RELEASE( indexBuffer );
                        HP_RELEASE( vertexBuffer );
                }
                std::vector<Vertex> vertices;
                std::vector<Index> indices;
                ID3D11Buffer* vertexBuffer;
                ID3D11Buffer* indexBuffer;
        };
        // [const][cop-c][cop-a][mov-c][mov-a]
        // [  +  ][  0  ][  0  ][  +  ][  +  ]
        struct Model
        {
                Model( )
                { }
                Model( const Model& ) = delete;
                Model( Model&&  m ) : meshes( std::move( m.meshes ) )
                { }
                Model operator = ( const Model& ) = delete;
                Model operator = ( Model&& m )
                {
                        return Model( std::move( m ) );
                }
                std::vector<Mesh> meshes;
        };
        /*}   }   }   }  }  }  } } } }}}} Functions {{{{ { { {  {  {  {   {   {   {*/

        ModelDef builtInModelDef( BuiltInModelDef&& m );
        ModelDef loadedModelDef( LoadedModelDef&& m );
        void addVertex_IO( Mesh& mesh, const Vertex vertex );
        void addIndex_IO( Mesh& mesh, const Index index );
        void setBuffers_IO( Renderer& renderer, Mesh& mesh );
```

```
        Maybe<Model> loadModelFromFile_IO( Renderer& renderer, const String& filename, const
float scale );
        Maybe<Model> cubeMesh_IO( Renderer& renderer, const FVec3& dimensions );
        namespace
        {
                void addMesh_IO( Model& meshes, Mesh&& mesh );
                Maybe<Model> loadModelFromFBXFile_IO( Renderer& renderer, const String&
filename, const float scale );
                void computeTangentsAndBinormals_IO( Vertex* pVerticies, UInt32 vertexCount,
UInt32* pIndices,
                        UInt32 indexCount );
                bool createBuffers_IO( Renderer& renderer, Mesh& mesh );
                Model cubeMesh( const FVec3& dimensions );
        }
}

#pragma once
#include "camera.hpp"
#include "directx.hpp"
#include "vertex.hpp"
#include "../adt/maybe.hpp"
#include "../window/window.hpp"
namespace hp_fp
{
        struct WindowConfig;
        struct Renderer
        {
                Renderer( D3D_DRIVER_TYPE driverType, D3D_FEATURE_LEVEL featureLevel,
                        const WindowConfig& windowConfig ) : driverType( driverType ),
                        featureLevel( featureLevel ), device( nullptr ), deviceContext(
nullptr ),
                        swapChain( nullptr ), renderTargetView( nullptr ),
                        depthStencilView( nullptr ), windowConfig( windowConfig )
                { }
                Renderer( const Renderer& ) = delete;
                Renderer( Renderer&& r ) : driverType( std::move( r.driverType ) ),
featureLevel( std::move( r.featureLevel ) ), device( std::move( r.device ) ), deviceContext(
std::move( r.deviceContext ) ),
                        swapChain( std::move( r.swapChain ) ), renderTargetView( std::move(
r.renderTargetView ) ), depthStencilView( std::move( r.depthStencilView ) ),
                        windowConfig( std::move( r.windowConfig ) )
                { }
                Renderer operator = ( const Renderer& ) = delete;
                Renderer operator = ( Renderer&& r )
                {
                        return Renderer{ std::move( r ) };
                }
                D3D_DRIVER_TYPE driverType;
                D3D_FEATURE_LEVEL featureLevel;
                ID3D11Device* device;
                ID3D11DeviceContext* deviceContext;
                IDXGISwapChain* swapChain;
                ID3D11RenderTargetView* renderTargetView;
                ID3D11DepthStencilView* depthStencilView;
                CameraBuffer cameraBuffer;
                WindowConfig windowConfig;
        };
        /*}   }   }   }  }  }  } } } }}}} Functions {{{{ { { {  {  {  {   {   {   {*/

        Maybe<Renderer> init_IO( WindowHandle windowHandle, const WindowConfig& windowConfig
);
        void preRender_IO( Renderer& renderer );
        void present_IO( Renderer& renderer );
        bool createVertexBuffer_IO( Renderer& renderer, ID3D11Buffer** vertexBuffer, UInt32
byteWidth,
                const Vertex* initData );
        bool createIndexBuffer_IO( Renderer& renderer, ID3D11Buffer** indexBuffer, UInt32
byteWidth,
                const Index* initData );
        void setVertexBuffers_IO( Renderer& renderer, ID3D11Buffer** vertexBuffer, UInt32*
stride,
                UInt32* offset );
        void setIndexBuffer_IO( Renderer& renderer, ID3D11Buffer** indexBuffer );
```

```
        void drawIndexed_IO( Renderer& renderer, UInt32 indexCount, UInt32
startIndexLocation,
                UInt32 baseVertexLocation );
}

#pragma once
#include "directx.hpp"
#include "../math/color.hpp"
#include "../math/vec2.hpp"
#include "../math/vec3.hpp"
namespace hp_fp
{
        struct Vertex
        {
                FVec3 position;
                Color color;
                FVec2 texCoord;
                FVec3 normal;
                FVec3 tangent;
                FVec3 binormal;
        };
        static const D3D11_INPUT_ELEMENT_DESC D3D11_LAYOUT[] =
        {
                {
                        "POSITION", // semantic name
                        0, // semantic index
                        DXGI_FORMAT_R32G32B32_FLOAT, // format
                        0, // input slot
                        0, // aligned byte offset
                        D3D11_INPUT_PER_VERTEX_DATA, // input slot class
                        0 // instance data step rate
                },

                {
                        "COLOR", // semantic name
                        0, // semantic index
                        DXGI_FORMAT_R32G32B32A32_FLOAT, // format
                        0, // input slot
                        12, // aligned byte offset
                        D3D11_INPUT_PER_VERTEX_DATA, // input slot class
                        0 // instance data step rate
                },

                {
                        "TEXCOORD", // semantic name
                        0, // semantic index
                        DXGI_FORMAT_R32G32_FLOAT, // format
                        0, // input slot
                        28, // aligned byte offset
                        D3D11_INPUT_PER_VERTEX_DATA, // input slot class
                        0 // instance data step rate
                },

                {
                        "NORMAL", // semantic name
                        0, // semantic index
                        DXGI_FORMAT_R32G32B32_FLOAT, // format
                        0, // input slot
                        36, // aligned byte offset
                        D3D11_INPUT_PER_VERTEX_DATA, // input slot class
                        0 // instance data step rate
                },

                {
                        "TANGENT", // semantic name
                        0, // semantic index
                        DXGI_FORMAT_R32G32B32_FLOAT, // format
                        0, // input slot
                        48, // aligned byte offset
                        D3D11_INPUT_PER_VERTEX_DATA, // input slot class
                        0 // instance data step rate
                },

                {
```

```
                        "BINORMAL", // semantic name
                        0, // semantic index
                        DXGI_FORMAT_R32G32B32_FLOAT, // format
                        0, // input slot
                        60, // aligned byte offset
                        D3D11_INPUT_PER_VERTEX_DATA, // input slot class
                        0 // instance data step rate
                },
        };
}

#pragma once
namespace hp_fp
{
        struct Color
        {
                float r, g, b, a;
                Color( const float r = 0, const float g = 0, const float b = 0,
                        const float a = 1.0f ) : r( r ), g( g ), b( b ), a( a )
                { }
                operator float*( )
                {
                        return &r;
                }
        };
}

#pragma once
#include "plane.hpp"
namespace hp_fp
{
        enum struct FrustumSides : UInt8
        {
                Near, Far, Top, Right, Bottom, Left, Count
        };
        struct Frustum
        {
                Plane planes[static_cast<UInt8>( FrustumSides::Count )];
                FVec3 nearClipVerts[4];
                FVec3 farClipVerts[4];
                float fieldOfView; // radians
                float aspectRatio; // width / height
                float nearClipDist;
                float farClipDist;
        };
        Frustum init( const float fieldOfView, const float aspectRatio, const float
nearClipDist,
                const float farClipDist );
        bool isInside( const Frustum& frustum, const FVec3& point, const float radius );
}

#pragma once
#include "vec3.hpp"
#include "vec4.hpp"
#include "quat.hpp"
namespace hp_fp
{
        struct Mat4x4
        {
        public:
                union
                {
                        struct
                        {
                                float _11, _12, _13, _14, _21, _22, _23, _24, _31, _32, _33,
_34, _41, _42, _43, _44;
                        };
                        float m[4][4];
                };
                Mat4x4( ) : Mat4x4( 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1 )
                { }
                Mat4x4( float m11, float m12, float m13, float m14, float m21, float m22,
float m23, float m24,
```

```
                        float m31, float m32, float m33, float m34, float m41, float m42,
float m43, float m44 )
                        : _11( m11 ), _12( m12 ), _13( m13 ), _14( m14 ), _21( m21 ), _22(
m22 ), _23( m23 ), _24( m24 ),
                        _31( m31 ), _32( m32 ), _33( m33 ), _34( m34 ), _41( m41 ), _42( m42
), _43( m43 ), _44( m44 )
                { }
                static const Mat4x4 identity;
                Mat4x4 operator * ( const Mat4x4& mat ) const
                {
                        Mat4x4 result;
                        for ( int i = 0; i < 4; i++ )
                        {
                                for ( int j = 0; j < 4; j++ )
                                {
                                        result.m[i][j] =
                                                m[i][0] * mat.m[0][j] +
                                                m[i][1] * mat.m[1][j] +
                                                m[i][2] * mat.m[2][j] +
                                                m[i][3] * mat.m[3][j];
                                }
                        }
                        return result;
                }
        };
        /*}   }   }   }  }  }  } } } }}}} Functions {{{{ { { {  {  {  {   {   {   {*/

        FVec3 pos( const Mat4x4& mat );
        float determinant( const Mat4x4& mat );
        Mat4x4 inverse( const Mat4x4& mat );
        Mat4x4 matrixPerspectiveFovLH( const float fieldOfView, const float aspectRatio,
                const float nearClipDist, const float farClipDist );
        Mat4x4 rotToMat4x4( const FQuat& rot );
        Mat4x4 sclToMat4x4( const FVec3& scl );
        Mat4x4 posToMat4x4( const FVec3& pos );
        Mat4x4 rotSclPosToMat4x4( const FQuat& rot, const FVec3& scl, const FVec3& pos );
}

#pragma once
#include "vec3.hpp"
namespace hp_fp
{
        struct Plane
        {
        public:
                float a, b, c, d;
                Plane( ) : a( 0.f ), b( 0.f ), c( 0.f ), d( 0.f )
                { }
                Plane( float a, float b, float c, float d ) : a( a ), b( b ), c( c ), d( d )
                { }
        };
        Plane init( const FVec3& p0, const FVec3& p1, const FVec3& p2 );
        Plane normalize( const Plane& plane );
        Plane planeFromPointNormal( const FVec3& point, const FVec3& normal );
        bool isInside( const Plane& plane, const FVec3& point, const float radius );
        float planeDotCoord( const Plane& plane, const FVec3& point );
}

#pragma once
#include "vec3.hpp"
namespace hp_fp
{
        template<typename A>
        struct Quat
        {
        public:
                A x, y, z, w;
                Quat( const A x, const A y, const A z, const A w ) : x( x ), y( y ), z( z ),
w( w )
                { };
                Quat( ) : x( 0.0f ), y( 0.0f ), z( 0.0f ), w( 1.0f )
                { };
                static const Quat<A> identity;
                Quat operator * ( const Quat& q ) const
```

```
		{
			return Quat{
				x * q.w + w * q.x + z * q.y - y * q.z,
				y * q.w - z * q.x + w * q.y + x * q.z,
				z * q.w + y * q.x - x * q.y + w * q.z,
				w * q.w - x * q.x - y * q.y - z * q.z };
		}
	};
	typedef Quat<float> FQuat;
	/*}   }   }   }  }  }  } } } }}}} Functions {{{{ { { {  {  {  {   {   {   {*/

	template<typename A>
	Quat<A> eulerDegToQuat( const Vec3<A>& vec )
	{
		return eulerRadToQuat( vec * DEG_TO_RAD_F );
	}
	template<typename A>
	Quat<A> eulerRadToQuat( const Vec3<A>& vec )
	{
		Quat<A> quat;
		quat.x = sin( vec.y / 2.0f ) * cos( vec.x / 2.0f ) * sin( vec.z / 2.0f ) +
			cos( vec.y / 2.0f ) * sin( vec.x / 2.0f ) * cos( vec.z / 2.0f );
		quat.y = sin( vec.y / 2.0f ) * cos( vec.x / 2.0f ) * cos( vec.z / 2.0f ) -
			cos( vec.y / 2.0f ) * sin( vec.x / 2.0f ) * sin( vec.z / 2.0f );
		quat.z = cos( vec.y / 2.0f ) * cos( vec.x / 2.0f ) * sin( vec.z / 2.0f ) -
			sin( vec.y / 2.0f ) * sin( vec.x / 2.0f ) * cos( vec.z / 2.0f );
		quat.w = cos( vec.y / 2.0f ) * cos( vec.x / 2.0f ) * cos( vec.z / 2.0f ) +
			sin( vec.y / 2.0f ) * sin( vec.x / 2.0f ) * sin( vec.z / 2.0f );
		return quat;
	}
	template<typename A>
	Quat<A> conjugate( const Quat<A>& quat )
	{
		return Quat < A > {-quat.x, -quat.y, -quat.z, quat.w};
	}
	template<typename A>
	Vec3<A> rotate( const Vec3<A>& vec, const Quat<A>& quat )
	{
		auto rotated = conjugate( quat ) * Quat < A > {vec.x, vec.y, vec.z, 1.0f}
*quat;
		return Vec3 < A > {rotated.x, rotated.y, rotated.z};
	};
}

#pragma once
namespace hp_fp
{
	template<typename A>
	struct Vec2
	{
	public:
		A x, y;
		/*Vec2( const A x = 0, const A y = 0 ) : x( x ), y( y )
		{ }*/
		Vec2<A> operator - ( const Vec2<A>& vec ) const
		{
			return Vec2 < A > { x - vec.x, y - vec.y };
		}
		bool operator == ( const Vec2<A>& v ) const
		{
			return x == v.x && y == v.y;
		}
		bool operator < ( const Vec2<A>& v ) const
		{
			if ( x == v.x )
			{
				return y < v.y;
			}
			return x < v.x;
		}
	};
	typedef Vec2<UInt16> UInt16Vec2;
	typedef Vec2<Int16> Int16Vec2;
	typedef Vec2<UInt32> UInt32Vec2;
```

```
        typedef Vec2<Int32> Int32Vec2;
        typedef Vec2<float> FVec2;
}

#pragma once
namespace hp_fp
{
        template<typename A>
        struct Vec3
        {
        public:
                A x, y, z;
                static const Vec3<A> zero;
                static const Vec3<A> right;
                static const Vec3<A> up;
                static const Vec3<A> forward;
                /*Vec3( const A x = 0, const A y = 0, const A z = 0 ) : x( x ), y( y ), z( z
)
                { }*/
                Vec3<A> operator + ( const Vec3<A>& vec ) const
                {
                        return Vec3 < A > { x + vec.x, y + vec.y, z + vec.z };
                }
                Vec3<A> operator - ( const Vec3<A>& vec ) const
                {
                        return Vec3 < A > { x - vec.x, y - vec.y, z - vec.z };
                }
                Vec3<A>& operator += ( const Vec3<A>& vec )
                {
                        x += vec.x;
                        y += vec.y;
                        z += vec.z;
                        return *this;
                }
                Vec3<A>& operator -= ( const Vec3<A>& vec )
                {
                        x -= vec.x;
                        y -= vec.y;
                        z -= vec.z;
                        return *this;
                }
                bool operator == ( const Vec3<A>& v ) const
                {
                        return x == v.x && y == v.y && z == v.z;
                }
                bool operator < ( const Vec3<A>& v ) const
                {
                        if ( x == v.x )
                        {
                                if ( y == v.y )
                                {
                                        return  z < v.z;
                                }
                                return y < v.y;
                        }
                        return x < v.x;
                }
                template<typename B>
                friend inline Vec3<B> operator * ( const float scalar, const Vec3<B>& vec );
                template<typename B>
                friend inline Vec3<B> operator * ( const Vec3<B>& vec, const float scalar );
                template<typename B>
                friend inline Vec3<B> operator / ( const Vec3<B>& vec, const float scalar );
        };
        typedef Vec3<UInt16> UInt16Vec3;
        typedef Vec3<Int16> Int16Vec3;
        typedef Vec3<UInt32> UInt32Vec3;
        typedef Vec3<Int32> Int32Vec3;
        typedef Vec3<float> FVec3;
        /*}   }   }   }  }  }  } } } }}}} Functions {{{{ { { {  {  {  {   {   {   {*/

        template<typename A>
        inline Vec3<A> operator * ( const float scalar, const Vec3<A>& vec )
        {
```

```
		return Vec3 < A > { vec.x * scalar, vec.y * scalar, vec.z * scalar };
	}
	template<typename A>
	inline Vec3<A> operator * ( const Vec3<A>& vec, const float scalar )
	{
		return scalar * vec;
	}
	template<typename A>
	inline Vec3<A> operator / ( const Vec3<A>& vec, const float scalar )
	{
		return Vec3 < A > { vec.x / scalar, vec.y / scalar, vec.z / scalar };
	}
	template<typename A>
	inline Vec3<A> cross( const Vec3<A>& vec1, const Vec3<A>& vec2 )
	{
		return Vec3 < A > {
			vec1.y * vec2.z - vec1.z * vec2.y,
				vec1.z * vec2.x - vec1.x * vec2.z,
				vec1.x * vec2.y - vec1.y * vec2.x };
	}
	template<typename A>
	inline A dot( const Vec3<A>& vec1, const Vec3<A>& vec2 )
	{
		return ( vec1.x ) * ( vec2.x ) + ( vec1.y ) * ( vec2.y ) + ( vec1.z ) * (
vec2.z );
	}
	template<typename A>
	Vec3<A> normalize( const Vec3<A>& vec )
	{
		Vec3<A> normVec{ };
		float len = length( vec );
		if ( len == 0.0f )
		{
			return normVec;
		}
		len = 1.0f / len;
		normVec.x = vec.x * len;
		normVec.y = vec.y * len;
		normVec.z = vec.z * len;
		return normVec;
	}
	template<typename A>
	float length( const Vec3<A>& vec )
	{
		return sqrtf( vec.x * vec.x + vec.y * vec.y + vec.z * vec.z );
	}
	template<typename A>
	float mag( const Vec3<A>& vec )
	{
		return length( vec );
	}
	template<typename A>
	Vec3<A> clampMag( const Vec3<A>& vec, const float max )
	{
		float magnitude = mag( vec );
		if ( magnitude > max )
		{
			return vec / magnitude * max;
		}
		return vec;
	}
}

#pragma once
#include "vec3.hpp"
namespace hp_fp
{
	template<typename A>
	struct Vec4
	{
	public:
		A x, y, z, w;
		static const Vec4<A> right;
		static const Vec4<A> up;
```

```
		static const Vec4<A> forward;
		Vec4( const A x = 0, const A y = 0, const A z = 0, const A w = 0 ) : x( x ),
y( y ), z( z ), w( w )
		{ }
		Vec4( const FVec3& vec );
		// not implemented yet
		static inline float Vec4Dot( const Vec4<A>* vec1, const Vec4<A>* vec2 )
		{
			return 0.f;
		}
		// not implemented yet
		static inline Vec3<A>* Vec4Normalize( Vec4<A>* vecOut, const Vec4<A>* vecIn )
		{
			return vecOut;
		}
		Vec4<A> operator + ( const Vec4<A>& vec ) const
		{
			return Vec4<A>( x + vec.x, y + vec.y, z + vec.z, 1.f );
		}
		Vec4<A> operator - ( const Vec4<A>& vec ) const
		{
			return Vec4<A>( x - vec.x, y - vec.y, z - vec.z, 1.f );
		}
		Vec4<A>& operator += ( const Vec4<A>& vec )
		{
			x += vec.x;
			y += vec.y;
			z += vec.z;
			return *this;
		}
		Vec4<A>& operator -= ( const Vec4<A>& vec )
		{
			x -= vec.x;
			y -= vec.y;
			z -= vec.z;
			return *this;
		}
		template<typename B >
		friend inline Vec4< B > operator * ( const float scalar, const Vec4< B >& vec
);
		template<typename B >
		friend inline Vec4< B > operator * ( const Vec4< B >& vec, const float scalar
);
	};
	template<typename A>
	inline Vec4<A> operator * ( const float scalar, const Vec4<A>& vec )
	{
		return Vec4<A>( vec.x * scalar, vec.y * scalar, vec.z * scalar, vec.w );
	}
	template<typename A>
	inline Vec4<A> operator * ( const Vec4<A>& vec, const float scalar )
	{
		return scalar * vec;
	}
	template<typename A>
	inline Vec4<A> cross( const Vec4<A>& vec1, const Vec4<A>& vec2, const Vec4<A>& vec3 )
	{
		return Vec4<A>(
			vec1.y * ( vec2.z * vec3.w - vec2.w * vec3.z )
			- vec1.z * ( vec2.y * vec3.w - vec2.w * vec3.y )
			+ vec1.w * ( vec2.y * vec3.z - vec2.z * vec3.y ),
			-vec1.x * ( vec2.z * vec3.w - vec2.w * vec3.z )
			+ vec1.z * ( vec2.x * vec3.w - vec2.w * vec3.x )
			- vec1.w * ( vec2.x * vec3.z - vec2.z * vec3.x ),
			vec1.x * ( vec2.y * vec3.w - vec2.w * vec3.y )
			- vec1.y * ( vec2.x * vec3.w - vec2.w * vec3.x )
			+ vec1.w * ( vec2.x * vec3.y - vec2.y * vec3.x ),
			-vec1.x * ( vec2.y * vec3.z - vec2.z * vec3.y )
			+ vec1.y * ( vec2.x * vec3.z - vec2.z * vec3.x )
			- vec1.z * ( vec2.x * vec3.y - vec2.y * vec3.x ) );
	}
	typedef Vec4<float> FVec4;
}
```

```
///
/// precompiled header
///
#pragma once

// disable STL expections
#define _HAS_EXCEPTIONS 0

#if defined( _WIN32 ) || defined( __WIN32__ )
// Windows
#   define HP_PLATFORM_WIN32
// exclude rarely-used services from Windows headers
#   ifndef WIN32_LEAN_AND_MEAN
#       define WIN32_LEAN_AND_MEAN
#   endif
#       ifndef NOMINMAX
#               define NOMINMAX
#       endif
#   include <Windows.h>
#else
#   error This operating system is not supported
#endif

// redefine new for debugging purposes
#if defined( _LOG )
#       define HP_NEW new( _NORMAL_BLOCK, __FILE__, __LINE__ )
#   define HP_DEBUG
#       include <iostream>
#       define ERR( x ) do { SetConsoleTextAttribute( GetStdHandle( STD_OUTPUT_HANDLE ), 0x0C
); std::cerr << __FILE__  << ":" << __LINE__ << ": " << x << std::endl; } while ( 0 )
#       define WAR( x ) do { SetConsoleTextAttribute( GetStdHandle( STD_OUTPUT_HANDLE ), 0x0E
); std::cerr << __FILE__  << ":" << __LINE__ << ": " << x << std::endl; } while ( 0 )
#       define LOG( x ) do { SetConsoleTextAttribute( GetStdHandle( STD_OUTPUT_HANDLE ), 0x07
); std::cout << __FILE__  << ":" << __LINE__ << ": " << x << std::endl; } while ( 0 )
#else
#       define HP_NEW new
#       define ERR( x )
#       define WAR( x )
#       define LOG( x )
#endif

// safe delete pointer
#ifndef HP_DELETE
#   define HP_DELETE( x ) delete x; x = nullptr;
#endif
// safe delete array
#ifndef HP_DELETE_ARRAY
#   define HP_DELETE_ARRAY( x ) delete [] x; x = nullptr;
#endif
// safe release
#ifndef HP_RELEASE
#   define HP_RELEASE( x ) if( x ) x->Release( ); x = nullptr;
#endif

#include <string>

#if defined( HP_PLATFORM_WIN32 )
typedef bool                            Bool;
typedef unsigned char           UInt8;
typedef signed char                     Int8;
typedef unsigned short          UInt16;
typedef signed short            Int16;
typedef unsigned int            UInt32;
typedef signed int                      Int32;
#   ifdef _MSC_VER
typedef signed __int64          Int64;
typedef unsigned __int64  UInt64;
#   else
typedef signed long long  Int64;
typedef unsigned long long      UInt64;
#   endif
typedef Int32                           Int;
typedef UInt32                          UInt;
typedef float                           Float;
```

```
typedef double                          Double;
typedef std::string                     String;
typedef HWND                            WindowHandle;
typedef UInt32                          Index;
#endif

namespace hp_fp
{
        extern const double PI;
        extern const double TWO_PI;
        extern const double DEG_TO_RAD;
        extern const double RAD_TO_DEG;
        extern const float PI_F;
        extern const float TWO_PI_F;
        extern const float DEG_TO_RAD_F;
        extern const float RAD_TO_DEG_F;
}

#pragma once
namespace hp_fp
{
#if defined( HP_PLATFORM_WIN32 )
        // converts multibyte string to unicode widechar string
        std::wstring s2ws( const std::string& s );
        // returns base path from a file path
        std::string basePath( const std::string& path );
#endif
}

#pragma once
namespace hp_fp
{
        typedef void* TypeId;
        template<typename A>
        // get type id without RTTI
        TypeId typeId( )
        {
                static A* typeUniqueMarker = NULL;
                return &typeUniqueMarker;
        }
}

#pragma once
#include <array>
namespace hp_fp
{
        enum struct Key : UInt8
        {
                Backspace = VK_BACK,
                Tab = VK_TAB,
                Return = VK_RETURN,
                Shift = VK_SHIFT,
                Ctrl = VK_CONTROL,
                Alt = VK_MENU,
                Pause = VK_PAUSE,
                CapsLock = VK_CAPITAL,
                Esc = VK_ESCAPE,
                Space = VK_SPACE,
                PageUp = VK_PRIOR,
                PageDown = VK_NEXT,
                End = VK_END,
                Home = VK_HOME,
                ArrowLeft = VK_LEFT,
                ArrowUp = VK_UP,
                ArrowRight = VK_RIGHT,
                ArrowDown = VK_DOWN,
                Print = VK_PRINT,
                PrintScreen = VK_SNAPSHOT,
                Insert = VK_INSERT,
                Delete = VK_DELETE,
                Help = VK_HELP,
                No0 = 0x30,
                No1 = 0x31,
                No2 = 0x32,
```

```
No3 = 0x33,
No4 = 0x34,
No5 = 0x35,
No6 = 0x36,
No7 = 0x37,
No8 = 0x38,
No9 = 0x39,
A = 'A',
B = 'B',
C = 'C',
D = 'D',
E = 'E',
F = 'F',
G = 'G',
H = 'H',
I = 'I',
J = 'J',
K = 'K',
L = 'L',
M = 'M',
N = 'N',
O = 'O',
P = 'P',
Q = 'Q',
R = 'R',
S = 'S',
T = 'T',
U = 'U',
V = 'V',
W = 'W',
X = 'X',
Y = 'Y',
Z = 'Z',
LWin = VK_LWIN,
RWin = VK_RWIN,
Num0 = VK_NUMPAD0,
Num1 = VK_NUMPAD1,
Num2 = VK_NUMPAD2,
Num3 = VK_NUMPAD3,
Num4 = VK_NUMPAD4,
Num5 = VK_NUMPAD5,
Num6 = VK_NUMPAD6,
Num7 = VK_NUMPAD7,
Num8 = VK_NUMPAD8,
Num9 = VK_NUMPAD9,
NumMultiply = VK_MULTIPLY,
NumAdd = VK_ADD,
NumSeparator = VK_SEPARATOR,
NumSubstract = VK_SUBTRACT,
NumDecimal = VK_DECIMAL,
NumDivide = VK_DIVIDE,
F1 = VK_F1,
F2 = VK_F2,
F3 = VK_F3,
F4 = VK_F4,
F5 = VK_F5,
F6 = VK_F6,
F7 = VK_F7,
F8 = VK_F8,
F9 = VK_F9,
F10 = VK_F10,
F11 = VK_F11,
F12 = VK_F12,
F13 = VK_F13,
F14 = VK_F14,
F15 = VK_F15,
F16 = VK_F16,
F17 = VK_F17,
F18 = VK_F18,
F19 = VK_F19,
F20 = VK_F20,
F21 = VK_F21,
F22 = VK_F22,
F23 = VK_F23,
```

```
        F24 = VK_F24,
        NumLock = VK_NUMLOCK,
        ScrollLock = VK_SCROLL,
        LShift = VK_LSHIFT,
        RShift = VK_RSHIFT,
        LCtrl = VK_LCONTROL,
        RCtrl = VK_RCONTROL,
        LAlt = VK_LMENU,
        RAlt = VK_RMENU
};
enum struct MouseButton : UInt8
{
        LeftButton = MK_LBUTTON,
        RightButton = MK_RBUTTON,
        MiddleButton = MK_MBUTTON,
        XButton1 = MK_XBUTTON1,
        XButton2 = MK_XBUTTON2
};
const size_t KEYS_SIZE = 255;
const size_t MOUSE_BUTTONS_SIZE = 5;
const size_t STATES_SIZE = KEYS_SIZE + MOUSE_BUTTONS_SIZE;
struct Mouse
{
        UInt16 x;
        UInt16 y;
        UInt16 delta; // wheel delta
};
// [const][cop-c][cop-a][mov-c][mov-a]
// [  +  ][  +  ][  +  ][  +  ][  +  ]
struct GameInput
{
        GameInput( )
        {
                std::fill_n( states.begin( ), STATES_SIZE, false );
        }
        GameInput( const GameInput& gi ) : states( gi.states ), mouse( gi.mouse ),
                text( gi.text )
        { }
        GameInput( GameInput&& gi ) : states( std::move( gi.states ) ),
                mouse( std::move( gi.mouse ) ), text( std::move( gi.text ) )
        { }
        GameInput operator = ( const GameInput& gi )
        {
                return GameInput{ gi };
        }
        GameInput operator = ( GameInput&& gi )
        {
                return GameInput{ std::move( gi ) };
        }
        bool& operator[]( const size_t i )
        {
                return states[i];
        }
        bool& operator[]( const Key k )
        {
                return states[static_cast<size_t>( k )];
        }
        bool& operator[]( const MouseButton k )
        {
                return states[KEYS_SIZE + static_cast<size_t>( k )];
        }
        bool operator[]( const Key k ) const
        {
                return states[static_cast<size_t>( k )];
        }
        bool operator[]( const MouseButton k ) const
        {
                return states[KEYS_SIZE + static_cast<size_t>( k )];
        }
private:
        GameInput( std::array<bool, STATES_SIZE> states, Mouse mouse, UInt32 text )
                : states( states ), mouse( mouse ), text( text )
        { }
        std::array<bool, STATES_SIZE> states;
```

```
        public:
                Mouse mouse;
                UInt32 text;
        };
}

#pragma once
#include "../adt/maybe.hpp"
namespace hp_fp
{
        enum struct WindowStyle : unsigned
        {
                Window,
                Fullscreen,
                Default = Window
        };
        struct WindowConfig
        {
                UInt width;
                UInt height;
                WindowStyle windowStyle;
                UInt bitsPerPx;
        };
        // [const][cop-c][cop-a][mov-c][mov-a]
        // [  +  ][  0  ][  0  ][  +  ][  +  ]
        struct Window
        {
                Window( WindowHandle handle, const LPCWSTR name ) : handle( handle ), name(
name )
                { }
                Window( const Window& ) = delete;
                Window( Window&& w ) : handle( std::move( w.handle ) ), name( std::move(
w.name ) )
                { }
                Window operator = ( const Window& ) = delete;
                Window operator = ( Window&& w )
                {
                        return Window{ std::move( w ) };
                }
                WindowHandle    handle;
                const LPCWSTR   name;
        };
        /*}   }   }   }  }  }  } } } }}}} Functions {{{{ { { {  {  {  {   {   {   {*/

        struct Engine;
        Maybe<Window> open_IO( Engine& engine, const WindowConfig& windowConfig );
        WindowConfig defaultWindowConfig_IO( );
        void setWindowVisibility_IO( WindowHandle windowHandle, const bool visible );
        void switchToFullscreen_IO( WindowHandle windowHandle, const WindowConfig&
windowConfig );
        void processMessages_IO( WindowHandle windowHandle );
        void captureMouse_IO( WindowHandle windowHandle );
        void releaseMouse_IO( );
        namespace
        {
                LRESULT CALLBACK windowProc_IO( WindowHandle handle, UINT message, WPARAM
wParam, LPARAM lParam );
                bool isOnlyInstance_IO( const LPCWSTR windowName );
        }
      }

#include <pch.hpp>
      #include "../../include/adt/frp/sfs.hpp"
      namespace hp_fp
      {
        SF<FVec3, FVec3> rotate( const S<FQuat>& rot )
        {
                return SF < FVec3, FVec3 >
                {
                        [rot]( const S<FVec3>& b ) -> S < FVec3 >
                        {
                                return S < FVec3 >
                                {
                                        [rot, b]( const float deltaMs ) -> FVec3
```

```
                                        {
                                                return rotate( b < deltaMs, rot < deltaMs );
                                        }
                                };
                        }
                };
        }
        SF<FVec3, FVec3> rotate( const FQuat& rot )
        {
                return SF < FVec3, FVec3 >
                {
                        [rot]( const S<FVec3>& b ) -> S < FVec3 >
                        {
                                return S < FVec3 >
                                {
                                        [rot, b]( const float deltaMs ) -> FVec3
                                        {
                                                return rotate( b < deltaMs, rot );
                                        }
                                };
                        }
                };
        }
      }

      #include <pch.hpp>
      #include "../../include/core/engine.hpp"
      #include <functional>
      #include "../../include/adt/maybe.hpp"
      #include "../../include/core/resources.hpp"
      #include "../../include/core/timer.hpp"
      #include "../../include/core/actor/actor.hpp"
      #include "../../include/graphics/renderer.hpp"
      #include "../../include/math/frustum.hpp"
      namespace hp_fp
      {
        Engine init( String&& name )
        {
                return Engine{ std::move( name ), EngineState::Initialized, { } };
        }
        void run_IO( Engine& engine, std::vector<ActorDef>&& actorDefs,
                const WindowConfig& windowConfig )
        {
                Maybe<Window> window = open_IO( engine, windowConfig );
                ifThenElse( window, [&engine, &windowConfig, &actorDefs]( Window& window )
                {
                        Maybe<Renderer> renderer = init_IO( window.handle, windowConfig );
                        ifThenElse( renderer, [&engine, &window, &actorDefs]( Renderer&
renderer )
                        {
                                engine.state = EngineState::Running;
                                Resources resources;
                                Timer timer = initTimer_IO( );
                                std::vector<Actor> actors = initActors_IO( renderer,
resources,
                                        std::move( actorDefs ) );
                                while ( engine.state == EngineState::Running )
                                {
                                        processMessages_IO( window.handle );
                                        updateTimer_IO( timer );
                                        preRender_IO( renderer );
                                        renderActors_IO( renderer, actors, engine.gameInput,
                                                static_cast<float>( timer.deltaMs ) );
                                        present_IO( renderer );
                                }
                        }, []
                        {
                                ERR( "Failed to initialize renderer." );
                        } );
                }, []
                {
                        ERR( "Failed to open a window." );
                } );
        }
```

```
        namespace
        {
                void renderActors_IO( Renderer& renderer, std::vector<Actor>& actors,
                        const GameInput& gameInput, const float deltaMs,
                        const Mat4x4& parentLocalTransform )
                {
                        for ( auto& actor : actors )
                        {
                                ActorInput actorInput{
                                        gameInput,
                                        actor.state
                                };
                                // render previous states first then run SF to be in sync
with cam
                                actor.render_IO( renderer, actor.state, parentLocalTransform
);
                                auto actorOutput = actor.sf < actorInput < deltaMs;
                                actor.state = actorOutput.state;
                                renderActors_IO( renderer, actor.children, gameInput,
deltaMs,
                                        trasformMatFromActorState( actor.state ) );
                        }
                }
                std::vector<Actor> initActors_IO( Renderer& renderer, Resources& resources,
                        std::vector<ActorDef>&& actorsDef )
                {
                        std::vector<Actor> actors{ };
                        actors.reserve( actorsDef.size( ) );
                        for ( auto& actorDef : actorsDef )
                        {
                                ActorState startingState{
                                        actorDef.startingState.pos,
                                        actorDef.startingState.vel,
                                        actorDef.startingState.scl,
                                        actorDef.startingState.rot,
                                        actorDef.startingState.modelRot
                                };
                                actors.push_back( Actor{ startingState, actorDef.sf,
                                        initActorRenderFunction_IO( renderer, resources,
actorDef ),
                                        initActors_IO( renderer, resources, std::move(
actorDef.children ) ) } );
                        }
                        return actors;
                }
        }
       }

       #include <pch.hpp>
       #include "../../include/core/resources.hpp"
       #include "../../include/core/actor/actor.hpp"
       namespace hp_fp
       {
         Maybe<Model>& getModel_IO( Renderer& renderer, Resources& resources,
                const LoadedModelDef& modelDef )
         {
                if ( resources.loadedModels.count( modelDef ) == 0 )
                {
                        resources.loadedModels.emplace( modelDef, loadModelFromFile_IO(
renderer,
                                modelDef.filename, modelDef.scale ) );
                }
                return resources.loadedModels.at( modelDef );
         }
         Maybe<Model>& getModel_IO( Renderer& renderer, Resources& resources,
                const BuiltInModelDef& modelDef )
         {
                if ( resources.builtInModels.count( modelDef ) == 0 )
                {
                        switch ( modelDef.type )
                        {
                        case BuiltInModelType::Cube:
                        {
                                resources.builtInModels.emplace( modelDef,
```

```
                                        cubeMesh_IO( renderer, modelDef.dimensions ) );
                        }
                        break;
                        default:
                                WAR( "Missing built-In model for type " +
                                        std::to_string( static_cast<UInt8>( modelDef.type )
) + "." );
                        }
                }
                return resources.builtInModels.at( modelDef );
        }
        Maybe<Material>& getMaterial_IO( Renderer& renderer, Resources& resources,
                const MaterialDef& materialDef )
        {
                if ( resources.materials.count( materialDef ) == 0 )
                {
                        resources.materials.emplace( materialDef, loadMaterial_IO( renderer,
materialDef ) );
                }
                return resources.materials.at( materialDef );
        }
        Maybe<ActorResources> getActorResources_IO( Renderer& renderer, Resources& resources,
                const ActorModelDef& actorModelDef )
        {
                if ( actorModelDef.model.is<LoadedModelDef>( ) )
                {
                        Maybe<Model>& model = getModel_IO( renderer, resources,
                                actorModelDef.model.loaded );
                        return getMaterialForModel_IO( renderer, resources, model,
                                actorModelDef.material );
                }
                else if ( actorModelDef.model.is<BuiltInModelDef>( ) )
                {
                        Maybe<Model>& model = getModel_IO( renderer, resources,
                                actorModelDef.model.builtIn );
                        return getMaterialForModel_IO( renderer, resources, model,
                                actorModelDef.material );
                }
                return nothing<ActorResources>( );
        }
        namespace
        {
                Maybe<ActorResources> getMaterialForModel_IO( Renderer& renderer, Resources&
resources,
                        Maybe<Model>& model, const MaterialDef& materialDef )
                {
                        return ifThenElse( model, [&renderer, &resources, &materialDef](
Model& model )
                        {
                                Maybe<Material>& material = getMaterial_IO( renderer,
resources, materialDef );
                                return ifThenElse( material, [&model]( Material& material )
                                {
                                        return just( ActorResources{ model, material } );
                                }, []
                                {
                                        return nothing<ActorResources>( );
                                } );
                        }, []
                        {
                                return nothing<ActorResources>( );
                        } );
                }
        }
      }

      #include <pch.hpp>
      #include "../../include/core/timer.hpp"
      namespace hp_fp
      {
        // http://randomascii.wordpress.com/2012/02/13/dont-store-that-in-a-float/
        // start timer at 2^32 to gain best precision
        const double TIME_ADDITION = 4294967296;
        Timer initTimer_IO( )
```

```
        {
                return Timer{ 0.0, getTimeMs_IO( ), 0.0 };
        }
        void updateTimer_IO( Timer& timer )
        {
                double currentTimeMs = getTimeMs_IO( );
                timer.deltaMs = currentTimeMs - timer._lastTimeMs;
                timer._lastTimeMs = currentTimeMs;
                timer._timeMs += timer.deltaMs;
        }
        double timeMs( const Timer& timer )
        {
                return timer._timeMs - TIME_ADDITION;
        }
        namespace
        {
                double getTimeMs_IO( )
                {
                        // use only single thread to calculate time
                        HANDLE currentThread = GetCurrentThread( );
                        DWORD_PTR previousMask = SetThreadAffinityMask( currentThread, 1 );
                        static LARGE_INTEGER frequency;
                        LARGE_INTEGER time;
                        QueryPerformanceFrequency( &frequency );
                        QueryPerformanceCounter( &time );
                        // use previously used thread
                        SetThreadAffinityMask( currentThread, previousMask );
                        return double( 1000 * time.QuadPart / frequency.QuadPart );
                }
        }
      }

      #include <pch.hpp>
      #include "../../include/core/actor/actor.hpp"
      #include "../../include/core/resources.hpp"
      namespace hp_fp
      {
        ActorTypeDef actorModelDef( ActorModelDef&& m )
        {
                ActorTypeDef def;
                def.model = m;
                def._typeId = typeId<ActorModelDef>( );
                return def;
        }
        ActorTypeDef actorCameraDef( ActorCameraDef&& c )
        {
                ActorTypeDef def;
                def.camera = c;
                def._typeId = typeId<ActorCameraDef>( );
                return def;
        }
        std::function<void( Renderer&, const ActorState&, const Mat4x4& )>
                initActorRenderFunction_IO( Renderer& renderer, Resources& resources,
                const ActorDef& actorDef )
        {
                static std::function<void( Renderer&, const ActorState&, const Mat4x4& )>
doNothing =
                        []( Renderer&, const ActorState&, const Mat4x4& )
                { };
                if ( actorDef.type.is<ActorModelDef>( ) )
                {
                        Maybe<ActorResources> res = getActorResources_IO( renderer,
resources,
                                actorDef.type.model );
                        return ifThenElse( res, []( ActorResources& res )
                        {
                                return renderActor_IO( res );
                        }, []
                        {
                                return doNothing;
                        } );
                }
                else if ( actorDef.type.is<ActorCameraDef>( ) )
                {
```

```
                        return actorDef.type.camera.render( actorDef.type.camera,
renderer.windowConfig );
                }
                return doNothing;
        }
        Mat4x4 trasformMatFromActorState( const ActorState& actorState )
        {
                return rotSclPosToMat4x4( actorState.rot, actorState.scl, actorState.pos );
        }
        Mat4x4 modelTrasformMatFromActorState( const ActorState& actorState )
        {
                return rotSclPosToMat4x4( actorState.modelRot * actorState.rot,
                        actorState.scl, actorState.pos );
        }
        namespace
        {
                // Have to specify lambda's return type to std::function because of the issue
                // with return type deduction
(http://stackoverflow.com/questions/12639578/c11-lambda-returning-lambda)
                std::function<void( Renderer&, const ActorState&, const Mat4x4& )>
renderActor_IO(
                        ActorResources& res )
                {
                        return [res]( Renderer& renderer, const ActorState& actorState,
                                const Mat4x4& transform ) mutable
                        {
                                const Camera& cam = getCamera( renderer.cameraBuffer );
                                setProjection_IO( res.material, cam.projection );
                                setView_IO( res.material, inverse( cam.transform ) );
                                setWorld_IO( res.material,
                                        modelTrasformMatFromActorState( actorState ) *
transform );
                                setCameraPosition_IO( res.material, pos( cam.transform ) );
                                setAbientLightColor_IO( res.material, Color( 0.1f, 0.1f,
0.1f, 0.6f ) );
                                setDiffuseLightColor_IO( res.material, Color( 1.0f, 0.95f,
0.4f, 0.4f ) );
                                setSpecularLightColor_IO( res.material, Color( 1.0f, 1.0f,
1.0f, 0.3f ) );
                                setLightDirection_IO( res.material, FVec3{ -0.5f, -1.0f,
0.1f } );
                                setTextures_IO( res.material );
                                setMaterials_IO( res.material );
                                bindInputLayout_IO( renderer, res.material );
                                for ( UInt32 i = 0; i < getPassCount( res.material ); ++i )
                                {
                                        applyPass_IO( renderer, res.material, i );
                                        for ( UInt32 j = 0; j < res.model.meshes.size( );
++j )
                                        {
                                                setBuffers_IO( renderer, res.model.meshes[j]
);
                                                drawIndexed_IO( renderer,
res.model.meshes[j].indices.size( ), 0, 0 );
                                        }
                                }
                        };
                };
        }
      }

      #include <pch.hpp>
      #include "../../include/graphics/camera.hpp"
      namespace hp_fp
      {
        const Camera& getCamera( const CameraBuffer& cameraBuffer )
        {
                if ( cameraBuffer._first )
                {
                        return cameraBuffer._cam[0];
                }
                else
                {
                        return cameraBuffer._cam[1];
```

```
                }
        }
        void setCamera_IO( CameraBuffer& cameraBuffer, Camera&& camera )
        {
                if ( cameraBuffer._first )
                {
                        cameraBuffer._cam[1] = camera;
                }
                else
                {
                        cameraBuffer._cam[0] = camera;
                }
        }
        void swap_IO( CameraBuffer& cameraBuffer )
        {
                cameraBuffer._first = !cameraBuffer._first;
        }
      }

      #include <pch.hpp>
      #include <algorithm>
      #include "../../include/graphics/material.hpp"
      #include "../../include/graphics/renderer.hpp"
      #include "../../include/graphics/vertex.hpp"
      #include "../../include/utils/string.hpp"
      namespace hp_fp
      {
        Material defaultMat( )
        {
                return Material
                {
                        "assets/shaders/parallax.fx", // filename
                        "Render", // techniqueName
                        { 1.0f, 1.0f }, // textureRepeat
                        { 0.5f, 0.5f, 0.5f }, // ambientMaterial
                        { 0.8f, 0.8f, 0.8f }, // diffuseMaterial
                        { 1.0f, 1.0f, 1.0f }, // specularMaterial
                        25.0f // specularPower
                };
        }
        Maybe<Material> loadMaterial_IO( Renderer& renderer, const MaterialDef& materialDef )
        {
                Material material = defaultMat( );
                bool materialLoaded;
                if ( materialLoaded = initMaterial_IO( material, renderer ) )
                {
                        if ( materialDef.diffuseTextureFilename != "" )
                        {
                                materialLoaded &= loadTexture_IO( &material.diffuseTexture,
renderer,
                                        materialDef.diffuseTextureFilename );
                        }
                        if ( materialDef.specularTextureFilename != "" )
                        {
                                materialLoaded &= loadTexture_IO( &material.specularTexture,
renderer,
                                        materialDef.specularTextureFilename );
                        }
                        if ( materialDef.bumpTextureFilename != "" )
                        {
                                materialLoaded &= loadTexture_IO( &material.bumpTexture,
renderer,
                                        materialDef.bumpTextureFilename );
                        }
                        if ( materialDef.parallaxTextureFilename != "" )
                        {
                                materialLoaded &= loadTexture_IO( &material.parallaxTexture,
renderer,
                                        materialDef.parallaxTextureFilename );
                        }
                        if ( materialDef.textureRepeat.x != 0.0f &&
materialDef.textureRepeat.y != 0.0f )
                        {
                                material.textureRepeat = materialDef.textureRepeat;
```

```
                        }
                }
                if ( materialLoaded )
                {
                        return just( std::move( material ) );
                }
                return nothing<Material>( );
        }
        bool loadTexture_IO( ID3D11ShaderResourceView** texture, Renderer& renderer,
                const String& filename )
        {
                String fileExt = filename.substr( filename.find( '.' ) + 1 );
                std::transform( fileExt.begin( ), fileExt.end( ), fileExt.begin( ), ::tolower
);
                std::wstring wFilename = s2ws( filename );
                if ( fileExt.compare( "dds" ) == 0 )
                {
                        if ( FAILED( DirectX::CreateDDSTextureFromFile( renderer.device,
                                wFilename.c_str( ), nullptr, texture ) ) )
                        {
                                ERR( "Failed to load \"" + filename + "\" texture." );
                                return false;
                        }
                }
                else
                {
                        if ( FAILED( DirectX::CreateWICTextureFromFile( renderer.device,
                                wFilename.c_str( ), nullptr, texture ) ) )
                        {
                                ERR( "Failed to load \"" + filename + "\" texture." );
                                return false;
                        }
                }
                return true;
        }
        namespace
        {
                bool initMaterial_IO( Material& material, Renderer& renderer )
                {
                        if ( loadShader_IO( material, renderer ) )
                        {
                                material.technique = material.effect->GetTechniqueByName(
material.techniqueName.c_str( ) );
                                material.technique->GetDesc( &material.techniqueDesc );
                                if ( createVertexLayout_IO( material, renderer ) )
                                {
                                        // retrieve all variables using semantic
                                        material.worldMatrixVariable = material.effect-
>GetVariableBySemantic( "WORLD" )->AsMatrix( );
                                        material.viewMatrixVariable = material.effect-
>GetVariableBySemantic( "VIEW" )->AsMatrix( );
                                        material.projectionMatrixVariable = material.effect-
>GetVariableBySemantic( "PROJECTION" )->AsMatrix( );
                                        material.diffuseTextureVariable = material.effect-
>GetVariableByName( "diffuseMap" )->AsShaderResource( );
                                        material.specularTextureVariable = material.effect-
>GetVariableByName( "specularMap" )->AsShaderResource( );
                                        material.bumpTextureVariable = material.effect-
>GetVariableByName( "bumpMap" )->AsShaderResource( );
                                        material.parallaxTextureVariable = material.effect-
>GetVariableByName( "heightMap" )->AsShaderResource( );
                                        material.envMapVariable = material.effect-
>GetVariableByName( "envMap" )->AsShaderResource( );
                                        // lights
                                        material.ambientLightColourVariable =
material.effect->GetVariableByName( "ambientLightColour" )->AsVector( );
                                        material.diffuseLightColourVariable =
material.effect->GetVariableByName( "diffuseLightColour" )->AsVector( );
                                        material.specularLightColourVariable =
material.effect->GetVariableByName( "specularLightColour" )->AsVector( );
                                        material.lightDirectionVariable = material.effect-
>GetVariableByName( "lightDirection" )->AsVector( );
                                        // materials
```

```
                                        material.ambientMaterialVariable = material.effect-
>GetVariableByName( "ambientMaterialColour" )->AsVector( );
                                        material.diffuseMaterialVariable = material.effect-
>GetVariableByName( "diffuseMaterialColour" )->AsVector( );
                                        material.specularMaterialVariable = material.effect-
>GetVariableByName( "specularMaterialColour" )->AsVector( );
                                        material.specularPowerVariable = material.effect-
>GetVariableByName( "specularPower" )->AsScalar( );
                                        material.textureRepeatVariable = material.effect-
>GetVariableByName( "textureRepeat" )->AsVector( );
                                        // camera
                                        material.cameraPositionVariable = material.effect-
>GetVariableByName( "cameraPosition" )->AsVector( );
                                        // booleans
                                        material.useDiffuseTextureVariable =
material.effect->GetVariableByName( "useDiffuseTexture" )->AsScalar( );
                                        material.useSpecularTextureVariable =
material.effect->GetVariableByName( "useSpecularTexture" )->AsScalar( );
                                        material.useBumpTextureVariable = material.effect-
>GetVariableByName( "useBumpTexture" )->AsScalar( );
                                        material.useParallaxTextureVariable =
material.effect->GetVariableByName( "useHeightTexture" )->AsScalar( );
                                        return true;
                                }
                        }
                        ERR( "Failed to initalize \"" + material.filename + "\"'s material."
);
                        return false;
                }
                bool loadShader_IO( Material& material, Renderer& renderer )
                {
                        ID3DBlob* buffer = NULL;
                        if ( !loadAndCompile_IO( material, renderer, "fx_5_0", &buffer ) )
                        {
                                return false;
                        }
                        if ( FAILED( D3DX11CreateEffectFromMemory( buffer->GetBufferPointer(
),
                                buffer->GetBufferSize( ), 0, renderer.device,
&material.effect ) ) )
                        {
                                HP_RELEASE( buffer );
                                return false;
                        }
                        return true;
                }
                bool loadAndCompile_IO( Material& material, Renderer& renderer, const String&
shaderModel,
                        ID3DBlob** buffer )
                {
                        std::wstring wFilePath = s2ws( material.filename );
                        DWORD shaderFlags = D3DCOMPILE_ENABLE_STRICTNESS;
        #       ifdef HP_DEBUG
                        shaderFlags |= D3DCOMPILE_DEBUG;
        #       endif
                        ID3DBlob* errorBuffer = 0;
                        HRESULT result = D3DCompileFromFile( wFilePath.c_str( ), 0, 0, 0,
shaderModel.c_str( ),
                                shaderFlags, 0, buffer, &errorBuffer );
                        if ( FAILED( result ) )
                        {
                                HP_RELEASE( errorBuffer );
                                return false;
                        }
                        HP_RELEASE( errorBuffer );
                        return true;
                }
                bool createVertexLayout_IO( Material& material, Renderer& renderer )
                {
                        UInt32 elementsCount = ARRAYSIZE( D3D11_LAYOUT );
                        D3DX11_PASS_DESC passDesc;
                        material.technique->GetPassByIndex( 0 )->GetDesc( &passDesc );
                        if ( FAILED( renderer.device->CreateInputLayout( D3D11_LAYOUT,
elementsCount,
```

```
                                passDesc.pIAInputSignature, passDesc.IAInputSignatureSize,
&material.inputLayout ) ) )
                        {
                                return false;
                        }
                        return true;
                }
        }
        void setProjection_IO( Material& material, const Mat4x4& mat )
        {
                material.projectionMatrixVariable->SetMatrix( (float*) ( &mat ) );
        }
        void setView_IO( Material& material, const Mat4x4& mat )
        {
                material.viewMatrixVariable->SetMatrix( (float*) ( &mat ) );
        }
        void setWorld_IO( Material& material, const Mat4x4& mat )
        {
                material.worldMatrixVariable->SetMatrix( (float*) ( &mat ) );
        }
        void setAbientLightColor_IO( Material& material, const Color& color )
        {
                material.ambientLightColourVariable->SetFloatVector( (float*) ( &color ) );
        }
        void setDiffuseLightColor_IO( Material& material, const Color& color )
        {
                material.diffuseLightColourVariable->SetFloatVector( (float*) ( &color ) );
        }
        void setSpecularLightColor_IO( Material& material, const Color& color )
        {
                material.specularLightColourVariable->SetFloatVector( (float*) ( &color ) );
        }
        void setLightDirection_IO( Material& material, const FVec3& dir )
        {
                material.lightDirectionVariable->SetFloatVector( (float*) ( &dir ) );
        }
        void setCameraPosition_IO( Material& material, const FVec3& dir )
        {
                material.cameraPositionVariable->SetFloatVector( (float*) ( &dir ) );
        }
        void setTextureRepeat_IO( Material& material, const FVec2& repeat )
        {
                material.textureRepeat = repeat;
        }
        void setTextures_IO( Material& material )
        {
                material.diffuseTextureVariable->SetResource( material.diffuseTexture );
                material.specularTextureVariable->SetResource( material.specularTexture );
                material.bumpTextureVariable->SetResource( material.bumpTexture );
                material.parallaxTextureVariable->SetResource( material.parallaxTexture );
                material.envMapVariable->SetResource( material.envMapTexture );

                if ( material.diffuseTexture )
                        material.useDiffuseTextureVariable->SetBool( true );
                if ( material.specularTexture )
                        material.useSpecularTextureVariable->SetBool( true );
                if ( material.bumpTexture )
                        material.useBumpTextureVariable->SetBool( true );
                if ( material.parallaxTexture )
                        material.useParallaxTextureVariable->SetBool( true );

                material.textureRepeatVariable->SetFloatVector( (float*) (
&material.textureRepeat ) );
        }
        void setMaterials_IO( Material& material )
        {
                material.ambientMaterialVariable->SetFloatVector( (float*) (
material.ambientMaterial ) );
                material.diffuseMaterialVariable->SetFloatVector( (float*) (
material.diffuseMaterial ) );
                material.specularMaterialVariable->SetFloatVector( (float*) (
material.specularMaterial ) );
                material.specularPowerVariable->SetFloat( material.specularPower );
        }
```

```
        void bindInputLayout_IO( Renderer& renderer, Material& material )
        {
                renderer.deviceContext->IASetInputLayout( material.inputLayout );
        }
        UInt32 getPassCount( Material& material )
        {
                return material.techniqueDesc.Passes;
        }
        void applyPass_IO( Renderer& renderer, Material& material, UInt32 i )
        {
                material.pass = material.technique->GetPassByIndex( i );
                material.pass->Apply( 0, renderer.deviceContext );
        }
      }

      #include <pch.hpp>
      #include <algorithm>
      #include <fbxsdk.h>
      #pragma comment(lib, "libfbxsdk-md.lib")
      #include "../../include/graphics/model.hpp"
      #include "../../include/graphics/renderer.hpp"
      namespace hp_fp
      {
        ModelDef builtInModelDef( BuiltInModelDef&& m )
        {
                ModelDef model;
                model.builtIn = m;
                model._typeId = typeId<BuiltInModelDef>( );
                return model;
        }
        ModelDef loadedModelDef( LoadedModelDef&& m )
        {
                ModelDef model;
                model.loaded = m;
                model._typeId = typeId<LoadedModelDef>( );
                return model;
        }
        void addVertex_IO( Mesh& mesh, const Vertex vertex )
        {
                mesh.vertices.push_back( vertex );
        }
        void addIndex_IO( Mesh& mesh, const Index index )
        {
                mesh.indices.push_back( index );
        }
        void setBuffers_IO( Renderer& renderer, Mesh& mesh )
        {
                UInt32 stride = sizeof( Vertex );
                UInt32 offset = 0;
                setVertexBuffers_IO( renderer, &mesh.vertexBuffer, &stride, &offset );
                setIndexBuffer_IO( renderer, &mesh.indexBuffer );
        }
        Maybe<Model> loadModelFromFile_IO( Renderer& renderer, const String& filename, const
float scale )
        {
                String fileExt{ filename.substr( filename.find( '.' ) + 1 ) };
                std::transform( fileExt.begin( ), fileExt.end( ), fileExt.begin( ), ::tolower
);
                if ( fileExt.compare( "fbx" ) == 0 )
                {
                        return loadModelFromFBXFile_IO( renderer, filename, scale );
                }
                return nothing<Model>( );
        }
        Maybe<Model> cubeMesh_IO( Renderer& renderer, const FVec3& dimensions )
        {
                Model model{ cubeMesh( dimensions ) };
                bool buffersCreated = true;
                for ( auto& mesh : model.meshes )
                {
                        buffersCreated &= createBuffers_IO( renderer, mesh );
                }
                if ( buffersCreated )
                {
```

```
                        return just( std::move( model ) );
                }
                ERR( "Failed to initalize cube's buffers." );
                return nothing<Model>( );
        }
        namespace
        {
                void addMesh_IO( Model& model, Mesh&& mesh )
                {
                        model.meshes.push_back( std::move( mesh ) );
                }
                Maybe<Model> loadModelFromFBXFile_IO( Renderer& renderer, const String&
filename, const float scale )
                {
                        Model model;
                        FbxManager* manager = FbxManager::Create( );
                        FbxIOSettings* settings = FbxIOSettings::Create( manager, IOSROOT );
                        manager->SetIOSettings( settings );
                        FbxImporter* importer = FbxImporter::Create( manager, "" );
                        FbxGeometryConverter converter( manager );
                        if ( !importer->Initialize( filename.c_str( ), -1, manager-
>GetIOSettings( ) ) )
                        {
                                return nothing<Model>( );
                        }
                        FbxScene* scene = FbxScene::Create( manager, "myScene" );
                        FbxAxisSystem axisSystem = scene->GetGlobalSettings(
).GetAxisSystem( );
                        importer->Import( scene );
                        importer->Destroy( );
                        FbxNode* rootNode = scene->GetRootNode( );
                        FbxMesh* fbxMesh = NULL;
                        if ( rootNode )
                        {
                                for ( Int32 i = 0; i < rootNode->GetChildCount( ); ++i )
                                {
                                        FbxNode* childNode = rootNode->GetChild( i );
                                        for ( Int32 j = 0; j < childNode-
>GetNodeAttributeCount( ); ++j )
                                        {
                                                FbxNodeAttribute* nodeAttribute = childNode-
>GetNodeAttributeByIndex( j );
                                                converter.Triangulate( nodeAttribute, true
);
                                                if ( nodeAttribute->GetAttributeType( ) ==
FbxNodeAttribute::eMesh )
                                                {
                                                        fbxMesh = (FbxMesh*) ( nodeAttribute
);
                                                        if ( fbxMesh )
                                                        {
                                                                Mesh mesh;
                                                                FbxVector4* fbxVertices =
fbxMesh->GetControlPoints( );
                                                                UInt32 vertexCount =
fbxMesh->GetControlPointsCount( );
                                                                UInt32 indexCount =
fbxMesh->GetPolygonVertexCount( );
                                                                Index* indices = (Index*)
fbxMesh->GetPolygonVertices( );
                                                                Vertex* vertices = new
Vertex[vertexCount];
                                                                for ( UInt32 k = 0; k <
vertexCount; ++k )
                                                                {

        vertices[k].position.x = float( fbxVertices[k][0] * scale );

        vertices[k].position.y = float( fbxVertices[k][1] * scale );

        vertices[k].position.z = float( fbxVertices[k][2] * scale );
                                                                }
```

```
                                                                for ( Int32 polyIndex = 0;
polyIndex < fbxMesh->GetPolygonCount( ); ++polyIndex )
                                                                {
                                                                        for ( UInt32
vertexIndex = 0; vertexIndex < 3; ++vertexIndex )
                                                                        {
                                                                                UInt32
cornerIndex = fbxMesh->GetPolygonVertex( polyIndex, vertexIndex );
                                                                                FbxVector4
fbxVertex = fbxVertices[cornerIndex];
                                                                                FbxVector4
fbxNormal;
                                                                                fbxMesh-
>GetPolygonVertexNormal( polyIndex, vertexIndex, fbxNormal );

        fbxNormal.Normalize( );

        vertices[cornerIndex].normal = FVec3{ float( fbxNormal[0] ),

        float( fbxNormal[1] ), float( fbxNormal[2] ) };
                                                                                FbxVector2
fbxUV = FbxVector2( 0, 0 );

        FbxLayerElementUV* fbxUVs = fbxMesh->GetLayer( 0 )->GetUVs( );
                                                                                if ( fbxUVs
)
                                                                                {

        UInt32 UVIndex = 0;

        switch ( fbxUVs->GetMappingMode( ) )
                                                                                        {
        case FbxLayerElement::eByControlPoint:

        UVIndex = cornerIndex;

        break;

        case FbxLayerElement::eByPolygonVertex:

        UVIndex = fbxMesh->GetTextureUVIndex( polyIndex, vertexIndex,
FbxLayerElement::eTextureDiffuse );

        break;
                                                                                        }

        fbxUV = fbxUVs->GetDirectArray( ).GetAt( UVIndex );

        vertices[cornerIndex].texCoord.x = float( fbxUV[0] );

        vertices[cornerIndex].texCoord.y = float( 1.0 - fbxUV[1] );
                                                                                }
                                                                        }
                                                                }

        computeTangentsAndBinormals_IO( vertices, vertexCount, indices, indexCount );
                                                                for ( UInt32 k = 0; k <
vertexCount; ++k )
                                                                {
                                                                        addVertex_IO( mesh,
vertices[k] );
                                                                }
                                                                for ( UInt32 k = 0; k <
indexCount; ++k )
                                                                {
                                                                        addIndex_IO( mesh,
indices[k] );
                                                                }
                                                                HP_DELETE_ARRAY( vertices
);
                                                                addMesh_IO( model,
std::move( mesh ) );
                                                        }
```

```
                                        }
                                }
                        }
                        bool buffersCreated = true;
                        for ( auto& mesh : model.meshes )
                        {
                                buffersCreated &= createBuffers_IO( renderer, mesh
);
                        }
                        if ( buffersCreated )
                        {
                                settings->Destroy( );
                                manager->Destroy( );
                                return just( std::move( model ) );
                        }
                        else
                        {
                                ERR( "Failed to initalize \"" + filename + "\"'s
buffers." );
                        }
                }
                ERR( "Failed to load \"" + filename + "\" model." );
                settings->Destroy( );
                manager->Destroy( );
                return nothing<Model>( );
        }
        void computeTangentsAndBinormals_IO( Vertex* vertices, UInt32 vertexCount,
UInt32* indices,
                UInt32 indexCount )
        {
                UInt32 triCount = indexCount / 3;
                FVec3* tangents = HP_NEW FVec3[vertexCount];
                FVec3* binormals = HP_NEW FVec3[vertexCount];
                for ( UInt32 i = 0; i < triCount; i += 3 )
                {
                        FVec3 v1 = vertices[indices[i]].position;
                        FVec3 v2 = vertices[indices[i + 1]].position;
                        FVec3 v3 = vertices[indices[i + 2]].position;
                        FVec2 uv1 = vertices[indices[i]].texCoord;
                        FVec2 uv2 = vertices[indices[i + 1]].texCoord;
                        FVec2 uv3 = vertices[indices[i + 2]].texCoord;
                        FVec3 edge1 = v2 - v1;
                        FVec3 edge2 = v3 - v1;
                        FVec2 edge1uv = uv2 - uv1;
                        FVec2 edge2uv = uv3 - uv1;
                        float cp = edge1uv.x * edge2uv.y - edge1uv.y * edge2uv.x;
                        if ( cp != 0.0f )
                        {
                                float mul = 1.0f / cp;
                                FVec3 tan = ( edge1 * edge2uv.y - edge2 * edge1uv.y
) * mul;
                                FVec3 binorm = ( edge1 * edge2uv.x - edge2 *
edge1uv.x ) * mul;
                                tangents[indices[i]] += tan;
                                binormals[indices[i]] += binorm;
                                tangents[indices[i + 1]] += tan;
                                binormals[indices[i + 1]] += binorm;
                                tangents[indices[i + 2]] += tan;
                                binormals[indices[i + 2]] += binorm;
                        }
                }
                for ( UInt32 i = 0; i < vertexCount; ++i )
                {
                        vertices[i].tangent = normalize( tangents[i] );
                        vertices[i].binormal = normalize( binormals[i] );
                }
                HP_DELETE_ARRAY( tangents );
                HP_DELETE_ARRAY( binormals );
        }
        bool createBuffers_IO( Renderer& renderer, Mesh& mesh )
        {
                if ( createVertexBuffer_IO( renderer, &mesh.vertexBuffer, sizeof(
Vertex ) * mesh.vertices.size( ), &mesh.vertices.at( 0 ) ) )
                {
```

```
                                if ( createIndexBuffer_IO( renderer, &mesh.indexBuffer,
sizeof( Index ) * mesh.indices.size( ), &mesh.indices.at( 0 ) ) )
                                {
                                        return true;
                                }
                        }
                        return false;
                }
                Model cubeMesh( const FVec3& dimensions )
                {
                        Model model;
                        Mesh mesh;
                        float halfWidth = dimensions.x / 2.f;
                        float halfHeight = dimensions.y / 2.f;
                        float halfLength = dimensions.z / 2.f;
                        Vertex vertices[] =
                        {
                                { FVec3{ -halfWidth, halfHeight, -halfLength }, Color( ),
FVec2{ 0.0f, 0.0f }, FVec3::up }, // 0 +Y (top face)
                                { FVec3{ halfWidth, halfHeight, -halfLength }, Color( ),
FVec2{ 1.0f, 0.0f }, FVec3::up }, // 1
                                { FVec3{ halfWidth, halfHeight, halfLength }, Color( ),
FVec2{ 1.0f, 1.0f }, FVec3::up }, // 2
                                { FVec3{ -halfWidth, halfHeight, halfLength }, Color( ),
FVec2{ 0.0f, 1.0f }, FVec3::up }, // 3

                                { FVec3{ -halfWidth, -halfHeight, halfLength }, Color( ),
FVec2{ 0.0f, 0.0f }, -1 * FVec3::up }, // 4 -Y (bottom face)
                                { FVec3{ halfWidth, -halfHeight, halfLength }, Color( ),
FVec2{ 1.0f, 0.0f }, -1 * FVec3::up }, // 5
                                { FVec3{ halfWidth, -halfHeight, -halfLength }, Color( ),
FVec2{ 1.0f, 1.0f }, -1 * FVec3::up }, // 6
                                { FVec3{ -halfWidth, -halfHeight, -halfLength }, Color( ),
FVec2{ 0.0f, 1.0f }, -1 * FVec3::up }, // 7

                                { FVec3{ halfWidth, halfHeight, halfLength }, Color( ),
FVec2{ 0.0f, 0.0f }, FVec3::right }, // 8 +X (right face)
                                { FVec3{ halfWidth, halfHeight, -halfLength }, Color( ),
FVec2{ 1.0f, 0.0f }, FVec3::right }, // 9
                                { FVec3{ halfWidth, -halfHeight, -halfLength }, Color( ),
FVec2{ 1.0f, 1.0f }, FVec3::right }, // 10
                                { FVec3{ halfWidth, -halfHeight, halfLength }, Color( ),
FVec2{ 0.0f, 1.0f }, FVec3::right }, // 11

                                { FVec3{ -halfWidth, halfHeight, -halfLength }, Color( ),
FVec2{ 0.0f, 0.0f }, -1 * FVec3::right }, // 12 -X (left face)
                                { FVec3{ -halfWidth, halfHeight, halfLength }, Color( ),
FVec2{ 1.0f, 0.0f }, -1 * FVec3::right }, // 13
                                { FVec3{ -halfWidth, -halfHeight, halfLength }, Color( ),
FVec2{ 1.0f, 1.0f }, -1 * FVec3::right }, // 14
                                { FVec3{ -halfWidth, -halfHeight, -halfLength }, Color( ),
FVec2{ 0.0f, 1.0f }, -1 * FVec3::right }, // 15

                                { FVec3{ -halfWidth, halfHeight, halfLength }, Color( ),
FVec2{ 0.0f, 0.0f }, FVec3::forward }, // 16 +Z (front face)
                                { FVec3{ halfWidth, halfHeight, halfLength }, Color( ),
FVec2{ 1.0f, 0.0f }, FVec3::forward }, // 17
                                { FVec3{ halfWidth, -halfHeight, halfLength }, Color( ),
FVec2{ 1.0f, 1.0f }, FVec3::forward }, // 18
                                { FVec3{ -halfWidth, -halfHeight, halfLength }, Color( ),
FVec2{ 0.0f, 1.0f }, FVec3::forward }, // 19

                                { FVec3{ halfWidth, halfHeight, -halfLength }, Color( ),
FVec2{ 0.0f, 0.0f }, -1 * FVec3::forward }, // 20 -Z (back face)
                                { FVec3{ -halfWidth, halfHeight, -halfLength }, Color( ),
FVec2{ 1.0f, 0.0f }, -1 * FVec3::forward }, // 21
                                { FVec3{ -halfWidth, -halfHeight, -halfLength }, Color( ),
FVec2{ 1.0f, 1.0f }, -1 * FVec3::forward }, // 22
                                { FVec3{ halfWidth, -halfHeight, -halfLength }, Color( ),
FVec2{ 0.0f, 1.0f }, -1 * FVec3::forward }, // 23
                        };
                        Index indices[] =
                        {
                                1, 0, 2, 2, 0, 3, // top face
```

```
                                5, 4, 6, 6, 4, 7, // bottom
                                9, 8, 10, 10, 8, 11, // right
                                13, 12, 14, 14, 12, 15, // left
                                17, 16, 18, 18, 16, 19, // front
                                21, 20, 22, 22, 20, 23 // back
                        };
                        computeTangentsAndBinormals_IO( vertices, 24, indices, 36 );
                        for ( UInt16 i = 0; i < 24; i++ )
                        {
                                addVertex_IO( mesh, vertices[i] );
                        }
                        for ( UInt16 i = 0; i < 36; i++ )
                        {
                                addIndex_IO( mesh, indices[i] );
                        }
                        addMesh_IO( model, std::move( mesh ) );
                        return model;
                }
          }
        }

        #include <pch.hpp>
        #include "../../include/graphics/renderer.hpp"
        namespace hp_fp
        {
          Maybe<Renderer> init_IO( WindowHandle windowHandle, const WindowConfig& windowConfig
)
          {
                Renderer renderer{ D3D_DRIVER_TYPE_NULL, D3D_FEATURE_LEVEL_11_0, windowConfig
};
                // driver types for fallback
                D3D_DRIVER_TYPE driverTypes[] = { D3D_DRIVER_TYPE_HARDWARE,
D3D_DRIVER_TYPE_WARP,
                        D3D_DRIVER_TYPE_SOFTWARE };
                UInt8 totalDriverTypes = ARRAYSIZE( driverTypes );
                // fallback feature levels
                D3D_FEATURE_LEVEL featureLevels[] = { D3D_FEATURE_LEVEL_11_0,
D3D_FEATURE_LEVEL_10_1,
                        D3D_FEATURE_LEVEL_10_0 };
                UInt8 totalFeatureLevels = ARRAYSIZE( featureLevels );
                // swap chain description
                DXGI_SWAP_CHAIN_DESC swapChainDesc;
                ZeroMemory( &swapChainDesc, sizeof( swapChainDesc ) );
                swapChainDesc.BufferCount = windowConfig.windowStyle ==
WindowStyle::Fullscreen ? 2 : 1;
                swapChainDesc.BufferDesc.Width = windowConfig.width;
                swapChainDesc.BufferDesc.Height = windowConfig.height;
                swapChainDesc.BufferDesc.Format = DXGI_FORMAT_R8G8B8A8_UNORM;
                swapChainDesc.BufferDesc.RefreshRate.Numerator = 60;
                swapChainDesc.BufferDesc.RefreshRate.Denominator = 1;
                swapChainDesc.BufferUsage = DXGI_USAGE_RENDER_TARGET_OUTPUT;
                swapChainDesc.OutputWindow = windowHandle;
                swapChainDesc.Windowed = windowConfig.windowStyle == WindowStyle::Window;
                swapChainDesc.SampleDesc.Count = 1;
                swapChainDesc.SampleDesc.Quality = 0;
                swapChainDesc.Flags = DXGI_SWAP_CHAIN_FLAG_ALLOW_MODE_SWITCH;
                // device creation flags
                UInt32 creationFlags = 0;
        #       ifdef HP_DEBUG
                creationFlags |= D3D11_CREATE_DEVICE_DEBUG;
        #       endif
                HRESULT result;
                UInt8 driver = 0;
                // loop through driver types and attempt to create device
                for ( driver; driver < totalDriverTypes; ++driver )
                {
                        result = D3D11CreateDeviceAndSwapChain( 0, driverTypes[driver], 0,
                                creationFlags, featureLevels, totalFeatureLevels,
D3D11_SDK_VERSION,
                                &swapChainDesc, &renderer.swapChain, &renderer.device,
                                &renderer.featureLevel, &renderer.deviceContext );
                        if ( SUCCEEDED( result ) )
                        {
                                renderer.driverType = driverTypes[driver];
```

```
                                break;
                        }
                }
                if ( FAILED( result ) )
                {
                        ERR( "Failed to create the Direct3D device!" );
                        return nothing<Renderer>( );
                }
                // back buffer texture to link render target with back buffer
                ID3D11Texture2D* backBufferTexture;
                if ( FAILED( renderer.swapChain->GetBuffer( 0, _uuidof( ID3D11Texture2D ),
                        (LPVOID*) &backBufferTexture ) ) )
                {
                        ERR( "Failed to get the swap chain back buffer!" );
                        return nothing<Renderer>( );
                }
                D3D11_TEXTURE2D_DESC depthDesc;
                depthDesc.Width = windowConfig.width;
                depthDesc.Height = windowConfig.height;
                depthDesc.MipLevels = 1;
                depthDesc.ArraySize = 1;
                depthDesc.Format = DXGI_FORMAT_D32_FLOAT;
                depthDesc.SampleDesc.Count = 1;
                depthDesc.SampleDesc.Quality = 0;
                depthDesc.Usage = D3D11_USAGE_DEFAULT;
                depthDesc.BindFlags = D3D11_BIND_DEPTH_STENCIL;
                depthDesc.CPUAccessFlags = 0;
                depthDesc.MiscFlags = 0;
                ID3D11Texture2D* depthStencilTexture;
                if ( FAILED( renderer.device->CreateTexture2D( &depthDesc, NULL,
                        &depthStencilTexture ) ) )
                {
                        ERR( "Failed to create depth stencil texture!" );
                        return nothing<Renderer>( );
                }
                D3D11_DEPTH_STENCIL_VIEW_DESC depthViewDesc;
                depthViewDesc.Format = depthDesc.Format;
                depthViewDesc.ViewDimension = D3D11_DSV_DIMENSION_TEXTURE2D;
                depthViewDesc.Texture2D.MipSlice = 0;
                depthViewDesc.Flags = 0;
                if ( FAILED( renderer.device->CreateDepthStencilView( depthStencilTexture,
                        &depthViewDesc, &renderer.depthStencilView ) ) )
                {
                        ERR( "Failed to create depth stencil view!" );
                        return nothing<Renderer>( );
                }
                if ( FAILED( renderer.device->CreateRenderTargetView( backBufferTexture,
NULL,
                        &renderer.renderTargetView ) ) )
                {
                        ERR( "Failed to create render target view!" );
                        HP_RELEASE( backBufferTexture );
                        return nothing<Renderer>( );
                }
                HP_RELEASE( backBufferTexture );
                renderer.deviceContext->OMSetRenderTargets( 1, &renderer.renderTargetView,
                        renderer.depthStencilView );
                // setup the viewport
                D3D11_VIEWPORT viewport;
                viewport.Width = static_cast<FLOAT>( windowConfig.width );
                viewport.Height = static_cast<FLOAT>( windowConfig.height );
                viewport.MinDepth = 0.0f;
                viewport.MaxDepth = 1.0f;
                viewport.TopLeftX = 0.f;
                viewport.TopLeftY = 0.f;
                renderer.deviceContext->RSSetViewports( 1, &viewport );
                renderer.deviceContext->IASetPrimitiveTopology(
D3D11_PRIMITIVE_TOPOLOGY_TRIANGLELIST );
                return just( std::move( renderer ) );
        }
        void preRender_IO( Renderer& renderer )
        {
                float ClearColor[4] = { 0.0f, 0.125f, 0.3f, 1.0f };
                renderer.deviceContext->ClearRenderTargetView( renderer.renderTargetView,
```

```
                        ClearColor );
                renderer.deviceContext->ClearDepthStencilView( renderer.depthStencilView,
                        D3D11_CLEAR_DEPTH, 1.0f, 0 );
        }
        void present_IO( Renderer& renderer )
        {
                swap_IO( renderer.cameraBuffer );
                renderer.swapChain->Present( 0, 0 );
        }
        bool createVertexBuffer_IO( Renderer& renderer, ID3D11Buffer** vertexBuffer,
                UInt32 byteWidth, const Vertex* initData )
        {
                D3D11_BUFFER_DESC bd;
                bd.Usage = D3D11_USAGE_DEFAULT;
                bd.ByteWidth = byteWidth;
                bd.BindFlags = D3D11_BIND_VERTEX_BUFFER;
                bd.CPUAccessFlags = 0;
                bd.MiscFlags = 0;
                D3D11_SUBRESOURCE_DATA subresourceData;
                subresourceData.pSysMem = initData;
                subresourceData.SysMemPitch = 0;
                subresourceData.SysMemSlicePitch = 0;
                if ( SUCCEEDED( renderer.device->CreateBuffer( &bd, &subresourceData,
vertexBuffer ) ) )
                {
                        return true;
                }
                return false;
        }
        bool createIndexBuffer_IO( Renderer& renderer, ID3D11Buffer** indexBuffer,
                UInt32 byteWidth, const Index* initData )
        {
                D3D11_BUFFER_DESC bd;
                bd.Usage = D3D11_USAGE_DEFAULT;
                bd.ByteWidth = byteWidth;
                bd.BindFlags = D3D11_BIND_INDEX_BUFFER;
                bd.CPUAccessFlags = 0;
                bd.MiscFlags = 0;
                D3D11_SUBRESOURCE_DATA subresourceData;
                subresourceData.pSysMem = initData;
                subresourceData.SysMemPitch = 0;
                subresourceData.SysMemSlicePitch = 0;
                if ( SUCCEEDED( renderer.device->CreateBuffer( &bd, &subresourceData,
indexBuffer ) ) )
                {
                        return true;
                }
                return false;
        }
        void setVertexBuffers_IO( Renderer& renderer, ID3D11Buffer** vertexBuffer, UInt32*
stride,
                UInt32* offset )
        {
                renderer.deviceContext->IASetVertexBuffers( 0, 1, vertexBuffer, stride,
offset );
        }
        void setIndexBuffer_IO( Renderer& renderer, ID3D11Buffer** indexBuffer )
        {
                renderer.deviceContext->IASetIndexBuffer( *indexBuffer, DXGI_FORMAT_R32_UINT,
0 );
        }
        void drawIndexed_IO( Renderer& renderer, UInt32 indexCount, UInt32
startIndexLocation,
                UInt32 baseVertexLocation )
        {
                renderer.deviceContext->DrawIndexed( indexCount, startIndexLocation,
baseVertexLocation );
        }
      }

      ///
      /// replace WinMain function with main on Windows
      ///
      #include <pch.hpp>
```

```
	#ifdef HP_PLATFORM_WIN32
	extern int main( int argc, char ** argv );

	int WINAPI WinMain( HINSTANCE, HINSTANCE, LPSTR, INT )
	{
	  return main( __argc, __argv );
	}
	#endif

	#include <pch.hpp>
	#include "../../include/math/frustum.hpp"
	namespace hp_fp
	{
	  Frustum init( const float fieldOfView, const float aspectRatio, const float
nearClipDist,
		const float farClipDist )
	  {
		Frustum frustum;
		frustum.fieldOfView = fieldOfView;
		frustum.aspectRatio = aspectRatio;
		frustum.nearClipDist = nearClipDist;
		frustum.farClipDist = farClipDist;
		float tanHalfFov = tan( frustum.fieldOfView / 2.f );
		FVec3 nearRight = ( frustum.nearClipDist * tanHalfFov ) * frustum.aspectRatio
* FVec3::right;
		FVec3 farRight = ( frustum.farClipDist * tanHalfFov ) * frustum.aspectRatio *
FVec3::right;
		FVec3 nearUp = ( frustum.nearClipDist * tanHalfFov ) * frustum.aspectRatio *
FVec3::up;
		FVec3 farUp = ( frustum.farClipDist * tanHalfFov ) * frustum.aspectRatio *
FVec3::up;
		frustum.nearClipVerts[0] = ( frustum.nearClipDist * FVec3::forward ) -
nearRight + nearUp;
		frustum.nearClipVerts[1] = ( frustum.nearClipDist * FVec3::forward ) +
nearRight + nearUp;
		frustum.nearClipVerts[2] = ( frustum.nearClipDist * FVec3::forward ) +
nearRight - nearUp;
		frustum.nearClipVerts[3] = ( frustum.nearClipDist * FVec3::forward ) -
nearRight - nearUp;
		frustum.farClipVerts[0] = ( frustum.farClipDist * FVec3::forward ) - farRight
+ farUp;
		frustum.farClipVerts[1] = ( frustum.farClipDist * FVec3::forward ) + farRight
+ farUp;
		frustum.farClipVerts[2] = ( frustum.farClipDist * FVec3::forward ) + farRight
- farUp;
		frustum.farClipVerts[3] = ( frustum.farClipDist * FVec3::forward ) - farRight
- farUp;
		FVec3 origin;
		frustum.planes[static_cast<UInt8>( FrustumSides::Near )] = init(
frustum.nearClipVerts[2], frustum.nearClipVerts[1], frustum.nearClipVerts[0] );
		frustum.planes[static_cast<UInt8>( FrustumSides::Far )] = init(
frustum.farClipVerts[0], frustum.farClipVerts[1], frustum.farClipVerts[2] );
		frustum.planes[static_cast<UInt8>( FrustumSides::Right )] = init(
frustum.farClipVerts[2], frustum.farClipVerts[1], origin );
		frustum.planes[static_cast<UInt8>( FrustumSides::Top )] = init(
frustum.farClipVerts[1], frustum.farClipVerts[0], origin );
		frustum.planes[static_cast<UInt8>( FrustumSides::Left )] = init(
frustum.farClipVerts[0], frustum.farClipVerts[3], origin );
		frustum.planes[static_cast<UInt8>( FrustumSides::Bottom )] = init(
frustum.farClipVerts[3], frustum.farClipVerts[2], origin );
		return frustum;
	  }
	  bool isInside( const Frustum& frustum, const FVec3& point, const float radius )
	  {
		for ( UInt8 i = 0; i < static_cast<UInt8>( FrustumSides::Count ); ++i )
		{
			if ( !isInside( frustum.planes[i], point, radius ) )
			{
				return false;
			}
		}
		return true;
	  }
	}
```

```
#include <pch.hpp>
#include "../../include/math/mat4x4.hpp"
namespace hp_fp
{
  const Mat4x4 Mat4x4::identity( Mat4x4( 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1
) );
  FVec3 pos( const Mat4x4& mat )
  {
          return FVec3{ mat.m[3][0], mat.m[3][1], mat.m[3][2] };
  }
  float determinant( const Mat4x4& mat )
  {
          FVec4 v1, v2, v3;
          v1.x = mat.m[0][0];
          v1.y = mat.m[1][0];
          v1.z = mat.m[2][0];
          v1.w = mat.m[3][0];
          v2.x = mat.m[0][1];
          v2.y = mat.m[1][1];
          v2.z = mat.m[2][1];
          v2.w = mat.m[3][1];
          v3.x = mat.m[0][2];
          v3.y = mat.m[1][2];
          v3.z = mat.m[2][2];
          v3.w = mat.m[3][2];
          FVec4 x = cross( v1, v2, v3 );
          return -( mat.m[0][3] * x.x + mat.m[1][3] * x.y + mat.m[2][3] * x.z +
                  mat.m[3][3] * x.w );
  }
  Mat4x4 inverse( const Mat4x4& mat )
  {
          Mat4x4 invMat;
          FVec4 vec[3];
          int i, j;
          float det = determinant( mat );
          for ( i = 0; i < 4; ++i )
          {
                  for ( j = 0; j < 4; ++j )
                  {
                          if ( i != j )
                          {
                                  int a = ( j <= i ) ? j : j - 1;
                                  vec[a].x = mat.m[j][0];
                                  vec[a].y = mat.m[j][1];
                                  vec[a].z = mat.m[j][2];
                                  vec[a].w = mat.m[j][3];
                          }
                  }
                  FVec4 x = cross( vec[0], vec[1], vec[2] );
                  float cofactor;
                  for ( j = 0; j < 4; ++j )
                  {
                          switch ( j )
                          {
                          case 0: cofactor = x.x; break;
                          case 1: cofactor = x.y; break;
                          case 2: cofactor = x.z; break;
                          case 3: cofactor = x.w; break;
                          }
                          invMat.m[j][i] = pow( -1.0f, i ) * cofactor / det;
                  }
          }
          return invMat;
  }
  Mat4x4 matrixPerspectiveFovLH( const float fieldOfView, const float aspectRatio,
          const float nearClipDist, const float farClipDist )
  {
          Mat4x4 perspective;
          float tanFov = tan( fieldOfView / 2.0f );
          perspective.m[0][0] = 1.0f / ( aspectRatio * tanFov );
          perspective.m[1][1] = 1.0f / tanFov;
          perspective.m[2][2] = farClipDist / ( farClipDist - nearClipDist );
          perspective.m[2][3] = 1.0f;
```

```
                perspective.m[3][2] = ( farClipDist *  -nearClipDist ) / ( farClipDist -
nearClipDist );
                perspective.m[3][3] = 0.0f;
                return perspective;
        }
        Mat4x4 rotToMat4x4( const FQuat& rot )
        {
                Mat4x4 mat;
                mat.m[0][0] = 1.0f - 2.0f * ( rot.y * rot.y + rot.z * rot.z );
                mat.m[0][1] = 2.0f * ( rot.x *rot.y + rot.z * rot.w );
                mat.m[0][2] = 2.0f * ( rot.x * rot.z - rot.y * rot.w );
                mat.m[1][0] = 2.0f * ( rot.x * rot.y - rot.z * rot.w );
                mat.m[1][1] = 1.0f - 2.0f * ( rot.x * rot.x + rot.z * rot.z );
                mat.m[1][2] = 2.0f * ( rot.y *rot.z + rot.x *rot.w );
                mat.m[2][0] = 2.0f * ( rot.x * rot.z + rot.y * rot.w );
                mat.m[2][1] = 2.0f * ( rot.y *rot.z - rot.x *rot.w );
                mat.m[2][2] = 1.0f - 2.0f * ( rot.x * rot.x + rot.y * rot.y );
                return mat;
        }
        Mat4x4 posToMat4x4( const FVec3& pos )
        {
                Mat4x4 mat;
                mat.m[3][0] = pos.x;
                mat.m[3][1] = pos.y;
                mat.m[3][2] = pos.z;
                return mat;
        }
        Mat4x4 sclToMat4x4( const FVec3& scl )
        {
                Mat4x4 mat;
                mat.m[0][0] = scl.x;
                mat.m[1][1] = scl.y;
                mat.m[2][2] = scl.z;
                return mat;
        }
        Mat4x4 rotSclPosToMat4x4( const FQuat& rot, const FVec3& scl, const FVec3& pos )
        {
                Mat4x4 mat;
                mat.m[0][0] = 1.0f - 2.0f * ( rot.y * rot.y + rot.z * rot.z );
                mat.m[0][1] = 2.0f * ( rot.x *rot.y + rot.z * rot.w );
                mat.m[0][2] = 2.0f * ( rot.x * rot.z - rot.y * rot.w );
                mat.m[1][0] = 2.0f * ( rot.x * rot.y - rot.z * rot.w );
                mat.m[1][1] = 1.0f - 2.0f * ( rot.x * rot.x + rot.z * rot.z );
                mat.m[1][2] = 2.0f * ( rot.y *rot.z + rot.x *rot.w );
                mat.m[2][0] = 2.0f * ( rot.x * rot.z + rot.y * rot.w );
                mat.m[2][1] = 2.0f * ( rot.y *rot.z - rot.x *rot.w );
                mat.m[2][2] = 1.0f - 2.0f * ( rot.x * rot.x + rot.y * rot.y );
                mat.m[3][0] = pos.x;
                mat.m[3][1] = pos.y;
                mat.m[3][2] = pos.z;
                mat.m[0][0] *= scl.x;
                mat.m[1][1] *= scl.y;
                mat.m[2][2] *= scl.z;
                return mat;

        }
      }

      #include <pch.hpp>
      #include "../../include/math/plane.hpp"
      namespace hp_fp
      {
        Plane init( const FVec3& p0, const FVec3& p1, const FVec3& p2 )
        {
                FVec3 edge1, edge2, normal;
                edge1 = p1 - p0;
                edge2 = p2 - p0;
                normal = normalize( cross( edge1, edge2 ) );
                return normalize( planeFromPointNormal( p0, normal ) );
        }
        Plane normalize( const Plane& plane )
        {
                Plane normPlane;
                float mag;
```

```
                mag = 1.0f / sqrt( plane.a * plane.a + plane.b * plane.b + plane.c * plane.c
);
                normPlane.a = plane.a * mag;
                normPlane.b = plane.b * mag;
                normPlane.c = plane.c * mag;
                normPlane.d = plane.d * mag;
                return normPlane;
        }
        Plane planeFromPointNormal( const FVec3& point, const FVec3& normal )
        {
                Plane plane;
                plane.a = normal.x;
                plane.b = normal.y;
                plane.c = normal.z;
                plane.d = dot( point, normal );
                return plane;
        }
        bool isInside( const Plane& plane, const FVec3& point, const float radius )
        {
                float distance;
                distance = planeDotCoord( plane, point );
                return distance >= -radius;
        }
        float planeDotCoord( const Plane& plane, const FVec3& point )
        {
                return plane.a * point.x + plane.b * point.y + plane.c * point.z + plane.d;
        }
      }

      #include <pch.hpp>
      #include "../../include/math/quat.hpp"
      namespace hp_fp
      {
        const FQuat FQuat::identity( FQuat( 0.0f, 0.0f, 0.0f, 1.0f ) );
      }

      #include <pch.hpp>
      #include "../../include/math/vec3.hpp"
      namespace hp_fp
      {
        const FVec3 FVec3::zero{ 0.f, 0.f, 0.f };
        const FVec3 FVec3::right{ 1.f, 0.f, 0.f };
        const FVec3 FVec3::up{ 0.f, 1.f, 0.f };
        const FVec3 FVec3::forward{ 0.f, 0.f, 1.f };
      }

      #include <pch.hpp>
      #include "../../include/math/vec4.hpp"
      namespace hp_fp
      {
        template<typename A>
        Vec4<A>::Vec4( const FVec3& vec ) : x( vec.x ), y( vec.y ), z( vec.z ), w( 1.f )
        { }
        const FVec4 FVec4::right( 1.f, 0.f, 0.f, 0.f );
        const FVec4 FVec4::up( 0.f, 1.f, 0.f, 0.f );
        const FVec4 FVec4::forward( 0.f, 0.f, 1.f, 0.f );
      }

      #include <pch.hpp>
      #include "../../include/utils/string.hpp"
      namespace hp_fp
      {
        std::wstring s2ws( const std::string& s )
        {
                int len;
                int slength = ( int ) s.length( ) + 1;
                len = MultiByteToWideChar( CP_ACP, 0, s.c_str( ), slength, 0, 0 );
                wchar_t* buf = new wchar_t[len];
                MultiByteToWideChar( CP_ACP, 0, s.c_str( ), slength, buf, len );
                std::wstring r( buf );
                delete[] buf;
                return r;
        }
        std::string basePath( const std::string& path )
```

```
        {
                size_t pos = path.find_last_of( "\\/" );
                return ( std::string::npos == pos ) ? "" : path.substr( 0, pos + 1 );
        }
      }

      #include <pch.hpp>
      #include "../../include/core/engine.hpp"
      #include "../../include/utils/string.hpp"
      #include "../../include/window/window.hpp"
      #include "../../include/adt/maybe.hpp"
      namespace hp_fp
      {
        Maybe<Window> open_IO( Engine& engine, const WindowConfig& windowConfig )
        {
                std::wstring windowNameW = s2ws( "HP_FP::" + engine.name );
                Window window{ nullptr, windowNameW.c_str( ) };
                if ( isOnlyInstance_IO( window.name ) )
                {
                        // window class details
                        WNDCLASSEX windowClass = { 0 };
                        windowClass.cbSize = sizeof( WNDCLASSEX );
                        windowClass.style = CS_VREDRAW | CS_HREDRAW;
                        windowClass.lpfnWndProc = &windowProc_IO;
                        windowClass.cbClsExtra = 0;
                        windowClass.cbWndExtra = 0;
                        windowClass.hInstance = GetModuleHandle( nullptr );
                        windowClass.hIcon = 0;
                        windowClass.hIconSm = 0;
                        windowClass.hCursor = 0;
                        windowClass.hbrBackground = 0;
                        windowClass.lpszMenuName = 0;
                        windowClass.lpszClassName = window.name;
                        // register the window
                        if ( RegisterClassEx( &windowClass ) )
                        {
                                // find position and size
                                HDC screenDC = GetDC( nullptr );
                                unsigned left = ( GetDeviceCaps( screenDC, HORZRES ) -
windowConfig.width ) / 2;
                                unsigned top = ( GetDeviceCaps( screenDC, VERTRES ) -
windowConfig.height ) / 2;
                                unsigned width = windowConfig.width;
                                unsigned height = windowConfig.height;
                                ReleaseDC( nullptr, screenDC );
                                // set the style of the window
                                DWORD style = WS_VISIBLE;
                                if ( windowConfig.windowStyle == WindowStyle::Window )
                                {
                                        style |= WS_CAPTION | WS_MINIMIZEBOX | WS_THICKFRAME
| WS_MAXIMIZEBOX | WS_SYSMENU;
                                        // adjust the window size with the borders etc.
                                        RECT rectangle = { 0, 0, windowConfig.width,
windowConfig.height };
                                        AdjustWindowRect( &rectangle, style, false );
                                        width = rectangle.right - rectangle.left;
                                        height = rectangle.bottom - rectangle.top;
                                }
                                // create the window
                                window.handle = CreateWindowEx( 0, window.name, window.name,
style, left, top, width, height, GetDesktopWindow( ), nullptr, GetModuleHandle( nullptr ),
&engine );
                                if ( window.handle == nullptr )
                                {
                                        ERR( GetLastError( ) );
                                        return nothing<Window>( );
                                }
                                if ( windowConfig.windowStyle == WindowStyle::Fullscreen )
                                {
                                        switchToFullscreen_IO( window.handle, windowConfig
);
                                }
                        }
                }
```

```
                return just( std::move( window ) );
        }
        WindowConfig defaultWindowConfig_IO( )
        {
                DEVMODE mode;
                mode.dmSize = sizeof( mode );
                EnumDisplaySettings( nullptr, ENUM_CURRENT_SETTINGS, &mode );
                return WindowConfig{ mode.dmPelsWidth, mode.dmPelsHeight,
WindowStyle::Default, mode.dmBitsPerPel };
        }
        void setWindowVisibility_IO( WindowHandle windowHandle, const bool visible )
        {
                ShowWindow( windowHandle, visible ? SW_SHOW : SW_HIDE );
                if ( visible )
                {
                        SetFocus( windowHandle );
                        SetForegroundWindow( windowHandle );
                        SetActiveWindow( windowHandle );
                }
        }
        void switchToFullscreen_IO( WindowHandle windowHandle, const WindowConfig&
windowConfig )
        {
                // set display settings
                DEVMODE devMode;
                devMode.dmSize = sizeof( devMode );
                devMode.dmPelsWidth = windowConfig.width;
                devMode.dmPelsHeight = windowConfig.height;
                devMode.dmBitsPerPel = windowConfig.bitsPerPx;
                devMode.dmFields = DM_PELSWIDTH | DM_PELSHEIGHT | DM_BITSPERPEL;
                // change default display device settings
                if ( ChangeDisplaySettings( &devMode, CDS_FULLSCREEN ) !=
DISP_CHANGE_SUCCESSFUL )
                {
                        return;
                }
                // set window style
                SetWindowLong( windowHandle, GWL_STYLE, WS_POPUP | WS_CLIPCHILDREN |
WS_CLIPSIBLINGS );
                // set extended window style
                SetWindowLong( windowHandle, GWL_EXSTYLE, WS_EX_APPWINDOW );
                // set window size, position and z-order
                SetWindowPos( windowHandle, HWND_TOP, 0, 0, windowConfig.width,
windowConfig.height, SWP_FRAMECHANGED );
                // show the window
                setWindowVisibility_IO( windowHandle, true );
        }
        void processMessages_IO( WindowHandle windowHandle )
        {
                MSG message;
                while ( PeekMessage( &message, windowHandle, 0, 0, PM_REMOVE ) )
                {
                        TranslateMessage( &message );
                        DispatchMessage( &message );
                }
        }
        void captureMouse_IO( WindowHandle windowHandle )
        {
                SetCapture( windowHandle );
        }
        void releaseMouse_IO( )
        {
                ReleaseCapture( );
        }
        namespace
        {
                LRESULT CALLBACK windowProc_IO( WindowHandle handle, UINT message, WPARAM
wParam, LPARAM lParam )
                {
                        Engine* engine = reinterpret_cast<Engine*>( GetWindowLongPtr(
handle, GWL_USERDATA ) );
                        switch ( message )
                        {
                        case WM_CREATE:
```

```
                        {
                                CREATESTRUCT *cs = reinterpret_cast<CREATESTRUCT*>( lParam
);
                                engine = reinterpret_cast<Engine*>( cs->lpCreateParams );
                                SetLastError( 0 );
                                if ( SetWindowLongPtr( handle, GWL_USERDATA,
reinterpret_cast<LONG_PTR>( engine ) ) == 0 )
                                {
                                        if ( GetLastError( ) != 0 )
                                        {
                                                ERR( "Unable to set window user data." );
                                                return FALSE;
                                        }
                                }
                                break;
                        }
                        case WM_CLOSE:
                        {
                                engine->state = EngineState::Terminated;
                                break;
                        }
                        case WM_ACTIVATEAPP:
                        {
                                break;
                        }
                        case WM_KEYDOWN:
                        case WM_SYSKEYDOWN:
                        {
                                engine->gameInput[static_cast<Key>( wParam )] = true;
                                break;
                        }
                        case WM_KEYUP:
                        case WM_SYSKEYUP:
                        {
                                engine->gameInput[static_cast<Key>( wParam )] = false;
                                break;
                        }
                        case WM_CHAR:
                        {
                                engine->gameInput.text = wParam;
                                break;
                        }
                        case WM_MOUSEMOVE:
                        {
                                engine->gameInput.mouse.x = LOWORD( lParam );
                                engine->gameInput.mouse.y = HIWORD( lParam );
                                break;
                        }
                        case WM_LBUTTONDOWN:
                        {
                                captureMouse_IO( handle );
                                engine->gameInput[MouseButton::LeftButton] = true;
                                break;
                        }
                        case WM_LBUTTONUP:
                        {
                                releaseMouse_IO( );
                                engine->gameInput[MouseButton::LeftButton] = false;
                                break;
                        }
                        case WM_RBUTTONDOWN:
                        {
                                captureMouse_IO( handle );
                                engine->gameInput[MouseButton::RightButton] = true;
                                break;
                        }
                        case WM_RBUTTONUP:
                        {
                                releaseMouse_IO( );
                                engine->gameInput[MouseButton::RightButton] = false;
                                break;
                        }
                        case WM_MBUTTONDOWN:
                        {
```

```
                                captureMouse_IO( handle );
                                engine->gameInput[MouseButton::MiddleButton] = true;
                                break;
                        }
                        case WM_MBUTTONUP:
                        {
                                releaseMouse_IO( );
                                engine->gameInput[MouseButton::MiddleButton] = false;
                                break;
                        }
                        case WM_MOUSEWHEEL:
                        {
                                engine->gameInput.mouse.delta = HIWORD( wParam ) / 120;
                                break;
                        }
                        case WM_XBUTTONDOWN:
                        {
                                captureMouse_IO( handle );
                                engine->gameInput[HIWORD( wParam ) == XBUTTON1 ?
MouseButton::XButton1 : MouseButton::XButton2] = true;
                                break;
                        }
                        case WM_XBUTTONUP:
                        {
                                releaseMouse_IO( );
                                engine->gameInput[HIWORD( wParam ) == XBUTTON1 ?
MouseButton::XButton1 : MouseButton::XButton2] = false;
                                break;
                        }
                        default:
                                return DefWindowProc( handle, message, wParam, lParam );
                        }
                        return FALSE;
                }
                bool isOnlyInstance_IO( const LPCWSTR windowName )
                {
                        HANDLE handle = CreateMutex( nullptr, true, windowName );
                        if ( GetLastError( ) != ERROR_SUCCESS )
                        {
                                WindowHandle windowHandle = FindWindow( windowName, nullptr
);
                                if ( windowHandle != nullptr )
                                {
                                        setWindowVisibility_IO( windowHandle, true );
                                        return false;
                                }
                        }
                        return true;
                }
        }
}
```

Appendix F: Source Code of Imperative Programming Version

```
#pragma once
#include <core/engine.hpp>
#include <core/actor/component/cameraComponent.hpp>
#include <core/actor/component/modelComponent.hpp>
#include <core/actor/component/transformComponent.hpp>

#pragma once
#include <vector>
#include "resources.hpp"
#include "actor/actor.hpp"
#include "../graphics/renderer.hpp"
#include "../window/gameInput.hpp"
#include "../window/window.hpp"
namespace hp_ip
{
	class Renderer;
	enum class EngineState : UInt8
	{
		Initialized,
		Running,
		Terminated
	};
	class Engine
	{
	public:
		Engine( String&& name, EngineState&& state )
			: _name( std::move( name ) ), _state( std::move( state ) ),
			_pWindow( nullptr )
		{ }
		Engine( String&& name ) : Engine( std::move( name ), EngineState::Initialized
)
		{ }
		~Engine( )
		{
			HP_DELETE( _pRenderer );
			HP_DELETE( _pWindow );
		}
		void run( const WindowConfig& windowConfig = Window::defaultWindowConfig( )
);
		void addActor( Actor&& actor );
	private:
		const String _name;
		EngineState _state;
		GameInput _gameInput;
		Resources _resources;
		Window* _pWindow;
		Renderer* _pRenderer;
		std::vector<Actor> _actors;
	};
}

#pragma once
#include <map>
#include <tuple>
#include "../graphics/model.hpp"
#include "../graphics/material.hpp"
namespace hp_ip
{
	class Renderer;
	class Resources
	{
	public:
		Resources( )
		{ }
		~Resources( );
		Resources( const Resources& ) = delete;
		void operator = ( const Resources& ) = delete;
		Model* getModel( Renderer* pRenderer, const LoadedModelDef& loadedModelDef );
		Model* getModel( Renderer* pRenderer, const BuiltInModelDef& builtInModelDef
);
```

```
                Material* getMaterial( Renderer* pRenderer, const MaterialDef& materialDef );
        private:
                std::map<LoadedModelDef, Model*> _loadedModels;
                std::map<BuiltInModelDef, Model*> _builtInModels;
                std::map<MaterialDef, Material*> _materials;
                std::map<String, ID3D11ShaderResourceView*> _textures;
        };
}

#pragma once
namespace hp_ip
{
        extern const double TIME_ADDITION;
        class Timer
        {
        public:
                Timer( )
                        : _lastTimeMs( getTimeMs( ) ), _timeMs( TIME_ADDITION )
                { }
                double update( );
        private:
                double getTimeMs( );
                double _timeMs;
                double _lastTimeMs;
        public:
                double timeMs( ) const
                {
                        return _timeMs - TIME_ADDITION;
                }
        };
}

#pragma once
#include <vector>
#include "component/iComponent.hpp"
#include "component/transformComponent.hpp"
#include "../../window/gameInput.hpp"
namespace hp_ip
{
        class Actor
        {
        public:
                //Actor( TransformComponent&& transform )
                //      //: _transformComponent( /*std::forward<TransformComponent>(*/
transform/* )*/ )
                //{
                //      _pTransformComponent = HP_NEW TransformComponent( transform );
                //}
                Actor( TransformComponent* pTransform )
                {
                        _pTransformComponent = pTransform;
                }
                //Actor( Actor&& actor ) : _transformComponent( /*std::move(*/
actor._transformComponent/* )*/ ),
                //      _components( std::move( actor._components ) ),
                //      _children( std::move( actor._children ) )
                //{
                //      actor._components.clear( );
                //      _transformComponent._owner = this;
                //      for ( auto& component : _components )
                //      {
                //              component->_owner = this;
                //      }
                //}
                ~Actor( )
                { }
                void init( Resources& resources, Renderer* pRenderer );
                void update( const float deltaMs, const GameInput& input );
                void render( Renderer* pRenderer );
                void addComponent( iComponent* component );
                void addComponent( TransformComponent* component );
        private:
                TransformComponent* _pTransformComponent;
                std::vector<iComponent*> _components;
```

```
                std::vector<Actor> _children;
        public:
                TransformComponent& transformComponent( )
                {
                        return *_pTransformComponent;
                }
                void addChild( Actor&& actor )
                {
                        _children.push_back( std::forward<Actor>( actor ) );
                }
        };
}

#pragma once
#include "iComponent.hpp"
#include "../../../graphics/camera.hpp"
#include "../../../math/frustum.hpp"
namespace hp_ip
{
        class CameraComponent : public iComponent
        {
        public:
                CameraComponent( CameraDef&& cameraDef )
                        : _cameraDef( std::forward<CameraDef>( cameraDef ) )
                { }
                virtual void vInit( Resources& resources, Renderer* pRenderer ) override;
                virtual void vUpdate( const float deltaMs, const GameInput& input ) override;
        protected:
                CameraDef _cameraDef;
                Frustum _frustum;
                Mat4x4 _projection;
                Renderer* _pRenderer;
        };
}

#pragma once
#include "../../../window/gameInput.hpp"
namespace hp_ip
{
        enum class ComponentType : UInt8
        {
                Transform, Other, Count
        };
        class Resources;
        class Renderer;
        class Actor;
        class iComponent
        {
        public:
                friend class Actor;
                iComponent( ComponentType type = ComponentType::Other ) : _owner( nullptr ),
                        _type( type )
                { }
                virtual ~iComponent( )
                { }
                virtual void vInit( Resources& resources, Renderer* pRenderer )
                { }
                virtual void vUpdate( const float deltaMs, const GameInput& input )
                { }
                virtual void vRender( Renderer* pRenderer )
                { }
        protected:
                Actor* _owner;
        private:
                ComponentType _type;
        };
}

#pragma once
#include "iComponent.hpp"
#include "../../../graphics/material.hpp"
#include "../../../graphics/model.hpp"
namespace hp_ip
{
```

```
        class ModelComponent : public iComponent
        {
        public:
                ModelComponent( String&& filename, const float scale, MaterialDef&
materialDef )
                        : _model( nullptr ), _type( ModelType::Loaded ),
                        _loadedModelDef( std::make_tuple( std::forward<String>( filename ),
scale ) ),
                        _materialDef( std::forward<MaterialDef>( materialDef ) )
                { }
                ModelComponent( const BuiltInModelType modelType, const FVec3& dimensions,
                        MaterialDef&& materialDef ) : _model( nullptr ), _type(
ModelType::BuiltIn ),
                        _builtInModelDef( std::make_tuple( modelType, dimensions ) ),
                        _materialDef( std::forward<MaterialDef>( materialDef ) )
                { }
                ~ModelComponent()
                {
                        _material = nullptr;
                        _model = nullptr;
                }
                virtual void vInit( Resources& resources, Renderer* pRenderer ) override;
                virtual void vRender( Renderer* pRenderer ) override;
        protected:
                Model* _model;
                Material* _material;
                ModelType _type;
                LoadedModelDef _loadedModelDef;
                BuiltInModelDef _builtInModelDef;
                MaterialDef _materialDef;
        };
}

#pragma once
#include "iComponent.hpp"
#include "../../../math/mat4x4.hpp"
#include "../../../math/quat.hpp"
namespace hp_ip
{
        class TransformComponent : public iComponent
        {
        public:
                TransformComponent( const FVec3& pos, const FVec3& vel, const FVec3& scl,
                        const FQuat& rot = FQuat::identity, const FQuat& modelRot =
FQuat::identity,
                        const Mat4x4& parentTransform = Mat4x4::identity )
                        : iComponent( ComponentType::Transform ), _pos( pos ), _vel( vel ),
                        _scl( scl ), _rot( rot ), _modelRot( modelRot ),
                        _parentTransform( parentTransform )
                { }
                TransformComponent( TransformComponent&& transform ) : TransformComponent(
                        std::move( transform._pos ), std::move( transform._vel ),
                        std::move( transform._scl ), std::move( transform._rot ),
                        std::move( transform._modelRot ), std::move(
transform._parentTransform ) )
                { }
                TransformComponent operator = ( TransformComponent&& transform )
                {
                        return TransformComponent( std::forward<TransformComponent>(
transform ) );
                }
                virtual void vUpdate( const float deltaMs, const GameInput& input );
                Mat4x4 transform( ) const;
                Mat4x4 modelTransform( ) const;
        protected:
                FVec3 _pos;
                FVec3 _vel;
                FVec3 _scl;
                FQuat _rot;
                FQuat _modelRot;
                Mat4x4 _parentTransform;
        public:
                FVec3 pos( ) const
                {
```

```
                        return _pos;
                }
                FVec3 vel( ) const
                {
                        return _vel;
                }
                FVec3 scl( ) const
                {
                        return _scl;
                }
                FQuat rot( ) const
                {
                        return _rot;
                }
                FQuat modelRot( ) const
                {
                        return _modelRot;
                }
                void setPos( const FVec3& pos )
                {
                        _pos = pos;
                }
                void setVel( const FVec3 vel )
                {
                        _vel = vel;
                }
                void setScl( const FVec3 scl )
                {
                        _scl = scl;
                }
                void setRot( const FQuat rot )
                {
                        _rot = rot;
                }
                void setModelRot( const FQuat modelRot )
                {
                        _modelRot = modelRot;
                }
                void setParentTransform( const Mat4x4& transform )
                {
                        _parentTransform = transform;
                }
        };
}

#pragma once
#include "../math/mat4x4.hpp"
namespace hp_ip
{
        struct CameraDef
        {
                float nearClipDist;
                float farClipDist;
        };
        struct Camera
        {
                Mat4x4 projection;
                Mat4x4 transform;
        };
        class CameraBuffer
        {
        public:
                const Camera& getCamera( );
                void setCamera( Camera&& camera );
                void swap( );
        private:
                bool _first;
                Camera _cam[2];
        };
}

#pragma once
#include <d3d11_2.h>
#pragma comment(lib, "d3d11.lib")
```

```
#include <D3Dcompiler.h>
#pragma comment(lib, "d3dcompiler.lib")
#include <Effects11/inc/d3dx11effect.h>
#pragma comment(lib, "Effects11.lib")
#include "DirectXTex/DDSTextureLoader/DDSTextureLoader.h"
#include "DirectXTex/WICTextureLoader/WICTextureLoader.h"

#pragma once
#include "directx.hpp"
#include "../math/color.hpp"
#include "../math/mat4x4.hpp"
#include "../math/vec2.hpp"
namespace hp_ip
{
	struct MaterialDef
	{
		const char* diffuseTextureFilename;
		const char* specularTextureFilename;
		const char* bumpTextureFilename;
		const char* parallaxTextureFilename;
		const char* evnMapTextureFilename;
		FVec2 textureRepeat;
		bool operator == ( const MaterialDef& m ) const
		{
			return diffuseTextureFilename == m.diffuseTextureFilename &&
				specularTextureFilename == m.specularTextureFilename &&
				bumpTextureFilename == m.bumpTextureFilename &&
				parallaxTextureFilename == m.parallaxTextureFilename &&
				evnMapTextureFilename == m.evnMapTextureFilename &&
				textureRepeat == m.textureRepeat;
		}
		bool operator < ( const MaterialDef& m ) const
		{
			if ( diffuseTextureFilename == m.diffuseTextureFilename )
			{
				if ( specularTextureFilename == m.specularTextureFilename )
				{
					if ( bumpTextureFilename == m.bumpTextureFilename )
					{
						if ( parallaxTextureFilename ==
m.parallaxTextureFilename )
						{
							if ( evnMapTextureFilename ==
m.evnMapTextureFilename )
							{
								return  textureRepeat <
m.textureRepeat;
							}
							return evnMapTextureFilename <
m.evnMapTextureFilename;
						}
						return parallaxTextureFilename <
m.parallaxTextureFilename;
					}
					return bumpTextureFilename < m.bumpTextureFilename;
				}
				return specularTextureFilename < m.specularTextureFilename;
			}
			return diffuseTextureFilename < m.diffuseTextureFilename;
		}
	};
	class Renderer;
	class Material
	{
	public:
		static Material* loadMaterial( Renderer* pRenderer, const MaterialDef&
materialDef );
		Material( const String& filename, const String& techniqueName,
			const FVec2& textureRepeat, const Color& ambientMaterial,
			const Color& diffuseMaterial, const Color& specularMaterial,
			const float specularPower );
		~Material( );
	private:
		bool init( Renderer* pRenderer );
```

```
                bool createVertexLayout( Renderer* pRenderer );
                bool loadDiffuseTexture( Renderer* pRenderer, const String& filename );
                bool loadSpecularTexture( Renderer* pRenderer, const String& filename );
                bool loadBumpTexture( Renderer* pRenderer, const String& filename );
                bool loadParallaxTexture( Renderer* pRenderer, const String& filename );
                bool loadEnvMapTexture( Renderer* pRenderer, const String& filename );
        public:
                void setProjection( const Mat4x4& mat )
                {
                        _pProjectionMatrixVariable->SetMatrix( (float*) ( &mat ) );
                }
                void setView( const Mat4x4& mat )
                {
                        _pViewMatrixVariable->SetMatrix( (float*) ( &mat ) );
                }
                void setWorld( const Mat4x4& mat )
                {
                        _pWorldMatrixVariable->SetMatrix( (float*) ( &mat ) );
                }
                void setAbientLightColor( const Color& color )
                {
                        _pAmbientLightColourVariable->SetFloatVector( (float*) ( &color ) );
                }
                void setDiffuseLightColor( const Color& color )
                {
                        _pDiffuseLightColourVariable->SetFloatVector( (float*) ( &color ) );
                }
                void setSpecularLightColor( const Color& color )
                {
                        _pSpecularLightColourVariable->SetFloatVector( (float*) ( &color )
);
                }
                void setLightDirection( const FVec3& dir )
                {
                        _pLightDirectionVariable->SetFloatVector( (float*) ( &dir ) );
                }
                void setCameraPosition( const FVec3& dir )
                {
                        _pCameraPositionVariable->SetFloatVector( (float*) ( &dir ) );
                }
                void setTextureRepeat( const FVec2& repeat )
                {
                        _textureRepeat = repeat;
                }
                void setTextures( )
                {
                        _pDiffuseTextureVariable->SetResource( _pDiffuseTexture );
                        _pSpecularTextureVariable->SetResource( _pSpecularTexture );
                        _pBumpTextureVariable->SetResource( _pBumpTexture );
                        _pParallaxTextureVariable->SetResource( _pParallaxTexture );
                        _pEnvMapVariable->SetResource( _pEnvMapTexture );
                        if ( _pDiffuseTexture )
                                _pUseDiffuseTextureVariable->SetBool( true );
                        if ( _pSpecularTexture )
                                _pUseSpecularTextureVariable->SetBool( true );
                        if ( _pBumpTexture )
                                _pUseBumpTextureVariable->SetBool( true );
                        if ( _pParallaxTexture )
                                _pUseParallaxTextureVariable->SetBool( true );
                        _pTextureRepeatVariable->SetFloatVector( (float*) ( &_textureRepeat
) );
                }
                void setMaterials( )
                {
                        _pAmbientMaterialVariable->SetFloatVector( (float*) (
_ambientMaterial ) );
                        _pDiffuseMaterialVariable->SetFloatVector( (float*) (
_diffuseMaterial ) );
                        _pSpecularMaterialVariable->SetFloatVector( (float*) (
_specularMaterial ) );
                        _pSpecularPowerVariable->SetFloat( _specularPower );
                }
                void bindInputLayout( Renderer* pRenderer );
                UInt32 getPassCount( )
```

```
		{
			return _techniqueDesc.Passes;
		}
		void applyPass( Renderer* pRenderer, UInt32 i );
	private:
		bool loadEffectFromFile( Renderer* pRenderer );
		bool compileD3DShader( const String& filePath, const String& shaderModel,
			ID3DBlob** ppBuffer );
		bool loadTexture( ID3D11ShaderResourceView** texture, Renderer* pRenderer,
			const String& filename );
		String _filename;
		String _techniqueName;
		// effect variables
		ID3DX11Effect* _pEffect;
		ID3DX11EffectTechnique* _pTechnique;
		ID3DX11EffectPass* _pPass;
		// input layout
		ID3D11InputLayout* _pInputLayout;
		// technique desc
		D3DX11_TECHNIQUE_DESC _techniqueDesc;
		// effect variables(constants)
		ID3DX11EffectMatrixVariable* _pViewMatrixVariable;
		ID3DX11EffectMatrixVariable* _pProjectionMatrixVariable;
		ID3DX11EffectMatrixVariable* _pWorldMatrixVariable;
		// Textures
		ID3DX11EffectShaderResourceVariable* _pDiffuseTextureVariable;
		ID3DX11EffectShaderResourceVariable* _pSpecularTextureVariable;
		ID3DX11EffectShaderResourceVariable* _pBumpTextureVariable;
		ID3DX11EffectShaderResourceVariable* _pParallaxTextureVariable;
		ID3DX11EffectShaderResourceVariable* _pEnvMapVariable;
		ID3D11ShaderResourceView* _pDiffuseTexture;
		ID3D11ShaderResourceView* _pSpecularTexture;
		ID3D11ShaderResourceView* _pBumpTexture;
		ID3D11ShaderResourceView* _pParallaxTexture;
		ID3D11ShaderResourceView* _pEnvMapTexture;
		FVec2 _textureRepeat;
		// Texture switches
		ID3DX11EffectScalarVariable* _pUseDiffuseTextureVariable;
		ID3DX11EffectScalarVariable* _pUseSpecularTextureVariable;
		ID3DX11EffectScalarVariable* _pUseBumpTextureVariable;
		ID3DX11EffectScalarVariable* _pUseParallaxTextureVariable;
		// Light
		ID3DX11EffectVectorVariable* _pAmbientLightColourVariable;
		ID3DX11EffectVectorVariable* _pDiffuseLightColourVariable;
		ID3DX11EffectVectorVariable* _pSpecularLightColourVariable;
		// Direction
		ID3DX11EffectVectorVariable* _pLightDirectionVariable;
		// Material
		ID3DX11EffectVectorVariable* _pAmbientMaterialVariable;
		ID3DX11EffectVectorVariable* _pDiffuseMaterialVariable;
		ID3DX11EffectVectorVariable* _pSpecularMaterialVariable;
		ID3DX11EffectScalarVariable* _pSpecularPowerVariable;
		ID3DX11EffectVectorVariable* _pTextureRepeatVariable;
		// Camera
		ID3DX11EffectVectorVariable* _pCameraPositionVariable;
		// Material colours
		Color _ambientMaterial;
		Color _diffuseMaterial;
		Color _specularMaterial;
		float _specularPower;
	};
}

#pragma once
#include <vector>
#include "vertex.hpp"
namespace hp_ip
{
	class Mesh
	{
		friend class Model;
		friend class Renderer;
	public:
		Mesh( ) : _vertexBuffer( nullptr ), _indexBuffer( nullptr )
```

```
                { }
                ~Mesh( )
                {
                        HP_RELEASE( _indexBuffer );
                        HP_RELEASE( _vertexBuffer );
                }
                size_t indicesSize( )
                {
                        return _indices.size( );
                }
                /*std::vector<Vertex>& vertices( )
                {
                return _vertices;
                }
                std::vector<Index>& indices( )
                {
                return _indices;
                }*/
        private:
                std::vector<Vertex> _vertices;
                std::vector<Index> _indices;
                ID3D11Buffer* _vertexBuffer;
                ID3D11Buffer* _indexBuffer;
        };
}

#pragma once
#include <vector>
#include <tuple>
#include "mesh.hpp"
namespace hp_ip
{
        class Renderer;
        enum class ModelType : UInt8
        {
                Loaded, BuiltIn, Count
        };
        enum struct BuiltInModelType : UInt8
        {
                Box, Count
        };
        typedef std::tuple<String, float> LoadedModelDef;
        typedef std::tuple<BuiltInModelType, FVec3> BuiltInModelDef;
        class Model
        {
        public:
                friend class ModelComponent;
                Model( );
                static Model* loadModelFromFile( Renderer* pRenderer,
                        const LoadedModelDef& loadedModelDef );
                static Model* loadModelFromFBXFile( Renderer* pRenderer, const String&
filename,
                        const float scale );
                static Model* cubeMesh( Renderer* pRenderer, const FVec3& dimensions );
        private:
                static void computeTangentsAndBinormals( Vertex* vertices, UInt32
vertexCount,
                        UInt32* indices, UInt32 indexCount );
                static void addVertex( Mesh& mesh, const Vertex vertex );
                static void addIndex( Mesh& mesh, const Index index );
                static bool createBuffers( Renderer* pRenderer, Mesh& mesh );
                static Model* cubeMesh( const FVec3& dimensions );
        private:
                std::vector<Mesh> _meshes;
        };
}

#pragma once
#include "camera.hpp"
#include "directx.hpp"
#include "vertex.hpp"
#include "../window/window.hpp"
namespace hp_ip
{
```

```
        class Mesh;
        class Renderer
        {
        public:
                Renderer( const WindowConfig& windowConfig ) : _driverType(
D3D_DRIVER_TYPE_NULL ),
                        _featureLevel( D3D_FEATURE_LEVEL_11_0 ), _pDevice( nullptr ),
                        _pDeviceContext( nullptr ), _pSwapChain( nullptr ),
_pRenderTargetView( nullptr ),
                        _pDepthStencilView( nullptr ), _pDepthStencilTexture( nullptr ),
                        _windowConfig( windowConfig )
                { }
                bool init( WindowHandle windowHandle );
                void preRender( );
                void swapCameras( );
                void present( );
                bool createVertexBuffer( ID3D11Buffer** vertexBuffer, UInt32 byteWidth,
                        const Vertex* initData );
                bool createIndexBuffer( ID3D11Buffer** indexBuffer, UInt32 byteWidth,
                        const Index* initData );
                void setVertexBuffers( ID3D11Buffer** vertexBuffer, UInt32* stride,
                        UInt32* offset );
                void setIndexBuffer( ID3D11Buffer** indexBuffer );
                void setBuffers( Mesh& mesh );
                void drawIndexed( UInt32 indexCount, UInt32 startIndexLocation,
                        UInt32 baseVertexLocation );
        private:
                D3D_DRIVER_TYPE _driverType;
                D3D_FEATURE_LEVEL _featureLevel;
                ID3D11Device* _pDevice;
                ID3D11DeviceContext* _pDeviceContext;
                IDXGISwapChain* _pSwapChain;
                ID3D11RenderTargetView* _pRenderTargetView;
                ID3D11DepthStencilView* _pDepthStencilView;
                ID3D11Texture2D* _pDepthStencilTexture;
                CameraBuffer _cameraBuffer;
                WindowConfig _windowConfig;
        public:
                ID3D11Device* device( )
                {
                        return _pDevice;
                }
                ID3D11DeviceContext* deviceContext( )
                {
                        return _pDeviceContext;
                }
                const WindowConfig& windowConfig( ) const
                {
                        return _windowConfig;
                }
                void setCamera( Camera&& camera )
                {
                        _cameraBuffer.setCamera( std::forward<Camera>( camera ) );
                }
                const Camera& getCamera( )
                {
                        return _cameraBuffer.getCamera( );
                }
        };
}

#pragma once
#include "directx.hpp"
#include "../math/color.hpp"
#include "../math/vec2.hpp"
#include "../math/vec3.hpp"
namespace hp_ip
{
        struct Vertex
        {
                FVec3 position;
                Color color;
                FVec2 texCoord;
                FVec3 normal;
```

```
		FVec3 tangent;
		FVec3 binormal;
	};
	static const D3D11_INPUT_ELEMENT_DESC D3D11_LAYOUT[] =
	{
		{
			"POSITION", // semantic name
			0, // semantic index
			DXGI_FORMAT_R32G32B32_FLOAT, // format
			0, // input slot
			0, // aligned byte offset
			D3D11_INPUT_PER_VERTEX_DATA, // input slot class
			0 // instance data step rate
		},

		{
			"COLOR", // semantic name
			0, // semantic index
			DXGI_FORMAT_R32G32B32A32_FLOAT, // format
			0, // input slot
			12, // aligned byte offset
			D3D11_INPUT_PER_VERTEX_DATA, // input slot class
			0 // instance data step rate
		},

		{
			"TEXCOORD", // semantic name
			0, // semantic index
			DXGI_FORMAT_R32G32_FLOAT, // format
			0, // input slot
			28, // aligned byte offset
			D3D11_INPUT_PER_VERTEX_DATA, // input slot class
			0 // instance data step rate
		},

		{
			"NORMAL", // semantic name
			0, // semantic index
			DXGI_FORMAT_R32G32B32_FLOAT, // format
			0, // input slot
			36, // aligned byte offset
			D3D11_INPUT_PER_VERTEX_DATA, // input slot class
			0 // instance data step rate
		},

		{
			"TANGENT", // semantic name
			0, // semantic index
			DXGI_FORMAT_R32G32B32_FLOAT, // format
			0, // input slot
			48, // aligned byte offset
			D3D11_INPUT_PER_VERTEX_DATA, // input slot class
			0 // instance data step rate
		},

		{
			"BINORMAL", // semantic name
			0, // semantic index
			DXGI_FORMAT_R32G32B32_FLOAT, // format
			0, // input slot
			60, // aligned byte offset
			D3D11_INPUT_PER_VERTEX_DATA, // input slot class
			0 // instance data step rate
		},
	};
}

#pragma once
namespace hp_ip
{
	struct Color
	{
		float r, g, b, a;
		Color( const float r = 0, const float g = 0, const float b = 0,
```

```
                        const float a = 1.0f ) : r( r ), g( g ), b( b ), a( a )
                { }
                operator float*( )
                {
                        return &r;
                }
        };
}

#pragma once
#include "plane.hpp"
namespace hp_ip
{
        enum struct FrustumSides : UInt8
        {
                Near, Far, Top, Right, Bottom, Left, Count
        };
        struct Frustum
        {
                Plane planes[static_cast<UInt8>( FrustumSides::Count )];
                FVec3 nearClipVerts[4];
                FVec3 farClipVerts[4];
                float fieldOfView; // radians
                float aspectRatio; // width / height
                float nearClipDist;
                float farClipDist;
        };
        Frustum init( const float fieldOfView, const float aspectRatio, const float
nearClipDist,
                const float farClipDist );
        bool isInside( const Frustum& frustum, const FVec3& point, const float radius );
}

#pragma once
#include "vec3.hpp"
#include "vec4.hpp"
#include "quat.hpp"
namespace hp_ip
{
        struct Mat4x4
        {
        public:
                union
                {
                        struct
                        {
                                float _11, _12, _13, _14, _21, _22, _23, _24, _31, _32, _33,
_34, _41, _42, _43, _44;
                        };
                        float m[4][4];
                };
                Mat4x4( ) : Mat4x4( 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1 )
                { }
                Mat4x4( float m11, float m12, float m13, float m14, float m21, float m22,
float m23, float m24,
                        float m31, float m32, float m33, float m34, float m41, float m42,
float m43, float m44 )
                        : _11( m11 ), _12( m12 ), _13( m13 ), _14( m14 ), _21( m21 ), _22(
m22 ), _23( m23 ), _24( m24 ),
                        _31( m31 ), _32( m32 ), _33( m33 ), _34( m34 ), _41( m41 ), _42( m42
), _43( m43 ), _44( m44 )
                { }
                static const Mat4x4 identity;
                Mat4x4 operator * ( const Mat4x4& mat ) const
                {
                        Mat4x4 result;
                        for ( int i = 0; i < 4; i++ )
                        {
                                for ( int j = 0; j < 4; j++ )
                                {
                                        result.m[i][j] =
                                                m[i][0] * mat.m[0][j] +
                                                m[i][1] * mat.m[1][j] +
                                                m[i][2] * mat.m[2][j] +
```

```
                                        m[i][3] * mat.m[3][j];
                                }
                        }
                        return result;
                }
        };
        /*}   }   }   }  }  }  } } } }}}} Functions {{{{ { { {  {  {  {   {   {   {*/

        FVec3 pos( const Mat4x4& mat );
        float determinant( const Mat4x4& mat );
        Mat4x4 inverse( const Mat4x4& mat );
        Mat4x4 matrixPerspectiveFovLH( const float fieldOfView, const float aspectRatio,
                const float nearClipDist, const float farClipDist );
        Mat4x4 rotToMat4x4( const FQuat& rot );
        Mat4x4 sclToMat4x4( const FVec3& scl );
        Mat4x4 posToMat4x4( const FVec3& pos );
        Mat4x4 rotSclPosToMat4x4( const FQuat& rot, const FVec3& scl, const FVec3& pos );
}

#pragma once
#include "vec3.hpp"
namespace hp_ip
{
        struct Plane
        {
        public:
                float a, b, c, d;
                Plane( ) : a( 0.f ), b( 0.f ), c( 0.f ), d( 0.f )
                { }
                Plane( float a, float b, float c, float d ) : a( a ), b( b ), c( c ), d( d )
                { }
        };
        Plane init( const FVec3& p0, const FVec3& p1, const FVec3& p2 );
        Plane normalize( const Plane& plane );
        Plane planeFromPointNormal( const FVec3& point, const FVec3& normal );
        bool isInside( const Plane& plane, const FVec3& point, const float radius );
        float planeDotCoord( const Plane& plane, const FVec3& point );
}

#pragma once
#include "vec3.hpp"
namespace hp_ip
{
        template<typename A>
        struct Quat
        {
        public:
                A x, y, z, w;
                Quat( const A x, const A y, const A z, const A w ) : x( x ), y( y ), z( z ),
w( w )
                { };
                Quat( ) : x( 0.0f ), y( 0.0f ), z( 0.0f ), w( 1.0f )
                { };
                static const Quat<A> identity;
                Quat operator * ( const Quat& q ) const
                {
                        return Quat{
                                x * q.w + w * q.x + z * q.y - y * q.z,
                                y * q.w - z * q.x + w * q.y + x * q.z,
                                z * q.w + y * q.x - x * q.y + w * q.z,
                                w * q.w - x * q.x - y * q.y - z * q.z };
                }
        };
        typedef Quat<float> FQuat;
        /*}   }   }   }  }  }  } } } }}}} Functions {{{{ { { {  {  {  {   {   {   {*/

        template<typename A>
        Quat<A> eulerDegToQuat( const Vec3<A>& vec )
        {
                return eulerRadToQuat( vec * DEG_TO_RAD_F );
        }
        template<typename A>
        Quat<A> eulerRadToQuat( const Vec3<A>& vec )
        {
```

```
                Quat<A> quat;
                quat.x = sin( vec.y / 2.0f ) * cos( vec.x / 2.0f ) * sin( vec.z / 2.0f ) +
                        cos( vec.y / 2.0f ) * sin( vec.x / 2.0f ) * cos( vec.z / 2.0f );
                quat.y = sin( vec.y / 2.0f ) * cos( vec.x / 2.0f ) * cos( vec.z / 2.0f ) -
                        cos( vec.y / 2.0f ) * sin( vec.x / 2.0f ) * sin( vec.z / 2.0f );
                quat.z = cos( vec.y / 2.0f ) * cos( vec.x / 2.0f ) * sin( vec.z / 2.0f ) -
                        sin( vec.y / 2.0f ) * sin( vec.x / 2.0f ) * cos( vec.z / 2.0f );
                quat.w = cos( vec.y / 2.0f ) * cos( vec.x / 2.0f ) * cos( vec.z / 2.0f ) +
                        sin( vec.y / 2.0f ) * sin( vec.x / 2.0f ) * sin( vec.z / 2.0f );
                return quat;
        }
        template<typename A>
        Quat<A> conjugate( const Quat<A>& quat )
        {
                return Quat < A > {-quat.x, -quat.y, -quat.z, quat.w};
        }
        template<typename A>
        Vec3<A> rotate( const Vec3<A>& vec, const Quat<A>& quat )
        {
                auto rotated = conjugate( quat ) * Quat < A > {vec.x, vec.y, vec.z, 1.0f}
*quat;
                return Vec3 < A > {rotated.x, rotated.y, rotated.z};
        };
}

#pragma once
namespace hp_ip
{
        template<typename A>
        struct Vec2
        {
        public:
                A x, y;
                /*Vec2( const A x = 0, const A y = 0 ) : x( x ), y( y )
                { }*/
                Vec2<A> operator - ( const Vec2<A>& vec ) const
                {
                        return Vec2 < A > { x - vec.x, y - vec.y };
                }
                bool operator == ( const Vec2<A>& v ) const
                {
                        return x == v.x && y == v.y;
                }
                bool operator < ( const Vec2<A>& v ) const
                {
                        if ( x == v.x )
                        {
                                return y < v.y;
                        }
                        return x < v.x;
                }
        };
        typedef Vec2<UInt16> UInt16Vec2;
        typedef Vec2<Int16> Int16Vec2;
        typedef Vec2<UInt32> UInt32Vec2;
        typedef Vec2<Int32> Int32Vec2;
        typedef Vec2<float> FVec2;
}

#pragma once
namespace hp_ip
{
        template<typename A>
        struct Vec3
        {
        public:
                A x, y, z;
                static const Vec3<A> zero;
                static const Vec3<A> right;
                static const Vec3<A> up;
                static const Vec3<A> forward;
                /*Vec3( const A x = 0, const A y = 0, const A z = 0 ) : x( x ), y( y ), z( z
)
                { }*/
```

```
static inline Vec3<A> normalize( const Vec3<A>& vec )
{
        Vec3<A> normVec{ };
        float length = Vec3<A>::length( vec );
        if ( length == 0.0f )
        {
                return normVec;
        }
        length = 1.0f / length;
        normVec.x = vec.x * length;
        normVec.y = vec.y * length;
        normVec.z = vec.z * length;
        return normVec;
}
static float length( const Vec3<A>& vec )
{
        return sqrtf( vec.x * vec.x + vec.y * vec.y + vec.z * vec.z );
}
Vec3<A> operator + ( const Vec3<A>& vec ) const
{
        return Vec3 < A > { x + vec.x, y + vec.y, z + vec.z };
}
Vec3<A> operator - ( const Vec3<A>& vec ) const
{
        return Vec3 < A > { x - vec.x, y - vec.y, z - vec.z };
}
Vec3<A>& operator += ( const Vec3<A>& vec )
{
        x += vec.x;
        y += vec.y;
        z += vec.z;
        return *this;
}
Vec3<A>& operator -= ( const Vec3<A>& vec )
{
        x -= vec.x;
        y -= vec.y;
        z -= vec.z;
        return *this;
}
bool operator == ( const Vec3<A>& v ) const
{
        return x == v.x && y == v.y && z == v.z;
}
bool operator < ( const Vec3<A>& v ) const
{
        if ( x == v.x )
        {
                if ( y == v.y )
                {
                        return  z < v.z;
                }
                return y < v.y;
        }
        return x < v.x;
}
float length(  )
{
        return sqrtf( x * x + y * y + z * z );
}
float mag( )
{
        return length();
}
Vec3<A>& clampMag( const float max )
{
        float magnitude = mag( );
        if ( magnitude > max )
        {
                float mult = max / magnitude;
                x *= mult;
                y *= mult;
                z *= mult;
        }
```

```
                        return *this;
                }
                template<typename B>
                friend inline Vec3<B> operator * ( const float scalar, const Vec3<B>& vec );
                template<typename B>
                friend inline Vec3<B> operator * ( const Vec3<B>& vec, const float scalar );
        };
        typedef Vec3<UInt16> UInt16Vec3;
        typedef Vec3<Int16> Int16Vec3;
        typedef Vec3<UInt32> UInt32Vec3;
        typedef Vec3<Int32> Int32Vec3;
        typedef Vec3<float> FVec3;
        /*}   }   }   }  }  }  } } } }}}} Functions {{{{ { { {  {  {  {   {   {   {*/

        template<typename A>
        inline Vec3<A> operator * ( const float scalar, const Vec3<A>& vec )
        {
                return Vec3 < A > { vec.x * scalar, vec.y * scalar, vec.z * scalar };
        }
        template<typename A>
        inline Vec3<A> operator * ( const Vec3<A>& vec, const float scalar )
        {
                return scalar * vec;
        }
        template<typename A>
        inline Vec3<A> cross( const Vec3<A>& vec1, const Vec3<A>& vec2 )
        {
                return Vec3 < A > {
                        vec1.y * vec2.z - vec1.z * vec2.y,
                                vec1.z * vec2.x - vec1.x * vec2.z,
                                vec1.x * vec2.y - vec1.y * vec2.x };
        }
        template<typename A>
        inline A dot( const Vec3<A>& vec1, const Vec3<A>& vec2 )
        {
                return ( vec1.x ) * ( vec2.x ) + ( vec1.y ) * ( vec2.y ) + ( vec1.z ) * (
vec2.z );
        }
}

#pragma once
#include "vec3.hpp"
namespace hp_ip
{
        template<typename A>
        struct Vec4
        {
        public:
                A x, y, z, w;
                static const Vec4<A> right;
                static const Vec4<A> up;
                static const Vec4<A> forward;
                Vec4( const A x = 0, const A y = 0, const A z = 0, const A w = 0 ) : x( x ),
y( y ), z( z ), w( w )
                { }
                Vec4( const FVec3& vec );
                // not implemented yet
                static inline float Vec4Dot( const Vec4<A>* vec1, const Vec4<A>* vec2 )
                {
                        return 0.f;
                }
                // not implemented yet
                static inline Vec3<A>* Vec4Normalize( Vec4<A>* vecOut, const Vec4<A>* vecIn )
                {
                        return vecOut;
                }
                Vec4<A> operator + ( const Vec4<A>& vec ) const
                {
                        return Vec4<A>( x + vec.x, y + vec.y, z + vec.z, 1.f );
                }
                Vec4<A> operator - ( const Vec4<A>& vec ) const
                {
                        return Vec4<A>( x - vec.x, y - vec.y, z - vec.z, 1.f );
                }
```

```
                Vec4<A>& operator += ( const Vec4<A>& vec )
                {
                        x += vec.x;
                        y += vec.y;
                        z += vec.z;
                        return *this;
                }
                Vec4<A>& operator -= ( const Vec4<A>& vec )
                {
                        x -= vec.x;
                        y -= vec.y;
                        z -= vec.z;
                        return *this;
                }
                template<typename B >
                friend inline Vec4< B > operator * ( const float scalar, const Vec4< B >& vec
);
                template<typename B >
                friend inline Vec4< B > operator * ( const Vec4< B >& vec, const float scalar
);
        };
        template<typename A>
        inline Vec4<A> operator * ( const float scalar, const Vec4<A>& vec )
        {
                return Vec4<A>( vec.x * scalar, vec.y * scalar, vec.z * scalar, vec.w );
        }
        template<typename A>
        inline Vec4<A> operator * ( const Vec4<A>& vec, const float scalar )
        {
                return scalar * vec;
        }
        template<typename A>
        inline Vec4<A> cross( const Vec4<A>& vec1, const Vec4<A>& vec2, const Vec4<A>& vec3 )
        {
                return Vec4<A>(
                        vec1.y * ( vec2.z * vec3.w - vec2.w * vec3.z )
                        - vec1.z * ( vec2.y * vec3.w - vec2.w * vec3.y )
                        + vec1.w * ( vec2.y * vec3.z - vec2.z * vec3.y ),
                        -vec1.x * ( vec2.z * vec3.w - vec2.w * vec3.z )
                        + vec1.z * ( vec2.x * vec3.w - vec2.w * vec3.x )
                        - vec1.w * ( vec2.x * vec3.z - vec2.z * vec3.x ),
                        vec1.x * ( vec2.y * vec3.w - vec2.w * vec3.y )
                        - vec1.y * ( vec2.x * vec3.w - vec2.w * vec3.x )
                        + vec1.w * ( vec2.x * vec3.y - vec2.y * vec3.x ),
                        -vec1.x * ( vec2.y * vec3.z - vec2.z * vec3.y )
                        + vec1.y * ( vec2.x * vec3.z - vec2.z * vec3.x )
                        - vec1.z * ( vec2.x * vec3.y - vec2.y * vec3.x ) );
        }
        typedef Vec4<float> FVec4;
}

///
/// precompiled header
///
#pragma once

// disable STL expections
#define _HAS_EXCEPTIONS 0

#if defined( _WIN32 ) || defined( __WIN32__ )
// Windows
#   define HP_PLATFORM_WIN32
// exclude rarely-used services from Windows headers
#   ifndef WIN32_LEAN_AND_MEAN
#       define WIN32_LEAN_AND_MEAN
#   endif
#       ifndef NOMINMAX
#               define NOMINMAX
#       endif
#   include <Windows.h>
#else
#   error This operating system is not supported
#endif
```

```
// redefine new for debugging purposes
#if defined( _LOG )
#       define HP_NEW new( _NORMAL_BLOCK, __FILE__, __LINE__ )
#   define HP_DEBUG
#       include <iostream>
#       define ERR( x ) do { SetConsoleTextAttribute( GetStdHandle( STD_OUTPUT_HANDLE ), 0x0C
); std::cerr << __FILE__  << ":" << __LINE__ << ": " << x << std::endl; } while ( 0 )
#       define WAR( x ) do { SetConsoleTextAttribute( GetStdHandle( STD_OUTPUT_HANDLE ), 0x0E
); std::cerr << __FILE__  << ":" << __LINE__ << ": " << x << std::endl; } while ( 0 )
#       define LOG( x ) do { SetConsoleTextAttribute( GetStdHandle( STD_OUTPUT_HANDLE ), 0x07
); std::cout << __FILE__  << ":" << __LINE__ << ": " << x << std::endl; } while ( 0 )
#else
#       define HP_NEW new
#       define ERR( x )
#       define WAR( x )
#       define LOG( x )
#endif

// safe delete pointer
#ifndef HP_DELETE
#   define HP_DELETE( x ) delete x; x = nullptr;
#endif
// safe delete array
#ifndef HP_DELETE_ARRAY
#   define HP_DELETE_ARRAY( x ) delete [] x; x = nullptr;
#endif
// safe release
#ifndef HP_RELEASE
#   define HP_RELEASE( x ) if( x ) x->Release( ); x = nullptr;
#endif

#include <string>

#if defined( HP_PLATFORM_WIN32 )
typedef unsigned char             UInt8;
typedef signed char                       Int8;
typedef unsigned short            UInt16;
typedef signed short              Int16;
typedef unsigned int              UInt32;
typedef signed int                        Int32;
#   ifdef _MSC_VER
typedef signed __int64            Int64;
typedef unsigned __int64  UInt64;
#   else
typedef signed long long  Int64;
typedef unsigned long long        UInt64;
#   endif
typedef Int32                             Int;
typedef UInt32                            UInt;

typedef HWND              WindowHandle;
typedef std::string       String;
typedef UInt32            Index;
#endif

namespace hp_ip
{
        extern const double PI;
        extern const double TWO_PI;
        extern const double DEG_TO_RAD;
        extern const double RAD_TO_DEG;
        extern const float PI_F;
        extern const float TWO_PI_F;
        extern const float DEG_TO_RAD_F;
        extern const float RAD_TO_DEG_F;
}

#pragma once
namespace hp_ip
{
#if defined( HP_PLATFORM_WIN32 )
        // converts multibyte string to unicode widechar string
        std::wstring s2ws( const std::string& s );
        // returns base path from a file path
```

```
        std::string basePath( const std::string& path );
#endif
}

#pragma once
namespace hp_ip
{
        typedef void* TypeId;
        template<typename A>
        // get type id without RTTI
        TypeId typeId( )
        {
                static A* typeUniqueMarker = NULL;
                return &typeUniqueMarker;
        }
}

#pragma once
#include <array>
namespace hp_ip
{
        enum class Key : UInt8
        {
                Backspace = VK_BACK,
                Tab = VK_TAB,
                Return = VK_RETURN,
                Shift = VK_SHIFT,
                Ctrl = VK_CONTROL,
                Alt = VK_MENU,
                Pause = VK_PAUSE,
                CapsLock = VK_CAPITAL,
                Esc = VK_ESCAPE,
                Space = VK_SPACE,
                PageUp = VK_PRIOR,
                PageDown = VK_NEXT,
                End = VK_END,
                Home = VK_HOME,
                ArrowLeft = VK_LEFT,
                ArrowUp = VK_UP,
                ArrowRight = VK_RIGHT,
                ArrowDown = VK_DOWN,
                Print = VK_PRINT,
                PrintScreen = VK_SNAPSHOT,
                Insert = VK_INSERT,
                Delete = VK_DELETE,
                Help = VK_HELP,
                No0 = 0x30,
                No1 = 0x31,
                No2 = 0x32,
                No3 = 0x33,
                No4 = 0x34,
                No5 = 0x35,
                No6 = 0x36,
                No7 = 0x37,
                No8 = 0x38,
                No9 = 0x39,
                A = 'A',
                B = 'B',
                C = 'C',
                D = 'D',
                E = 'E',
                F = 'F',
                G = 'G',
                H = 'H',
                I = 'I',
                J = 'J',
                K = 'K',
                L = 'L',
                M = 'M',
                N = 'N',
                O = 'O',
                P = 'P',
                Q = 'Q',
                R = 'R',
```

```
        S = 'S',
        T = 'T',
        U = 'U',
        V = 'V',
        W = 'W',
        X = 'X',
        Y = 'Y',
        Z = 'Z',
        LWin = VK_LWIN,
        RWin = VK_RWIN,
        Num0 = VK_NUMPAD0,
        Num1 = VK_NUMPAD1,
        Num2 = VK_NUMPAD2,
        Num3 = VK_NUMPAD3,
        Num4 = VK_NUMPAD4,
        Num5 = VK_NUMPAD5,
        Num6 = VK_NUMPAD6,
        Num7 = VK_NUMPAD7,
        Num8 = VK_NUMPAD8,
        Num9 = VK_NUMPAD9,
        NumMultiply = VK_MULTIPLY,
        NumAdd = VK_ADD,
        NumSeparator = VK_SEPARATOR,
        NumSubstract = VK_SUBTRACT,
        NumDecimal = VK_DECIMAL,
        NumDivide = VK_DIVIDE,
        F1 = VK_F1,
        F2 = VK_F2,
        F3 = VK_F3,
        F4 = VK_F4,
        F5 = VK_F5,
        F6 = VK_F6,
        F7 = VK_F7,
        F8 = VK_F8,
        F9 = VK_F9,
        F10 = VK_F10,
        F11 = VK_F11,
        F12 = VK_F12,
        F13 = VK_F13,
        F14 = VK_F14,
        F15 = VK_F15,
        F16 = VK_F16,
        F17 = VK_F17,
        F18 = VK_F18,
        F19 = VK_F19,
        F20 = VK_F20,
        F21 = VK_F21,
        F22 = VK_F22,
        F23 = VK_F23,
        F24 = VK_F24,
        NumLock = VK_NUMLOCK,
        ScrollLock = VK_SCROLL,
        LShift = VK_LSHIFT,
        RShift = VK_RSHIFT,
        LCtrl = VK_LCONTROL,
        RCtrl = VK_RCONTROL,
        LAlt = VK_LMENU,
        RAlt = VK_RMENU
};
enum class MouseButton : UInt8
{
        LeftButton = MK_LBUTTON,
        RightButton = MK_RBUTTON,
        Shift = MK_SHIFT,
        Control = MK_CONTROL,
        MiddleButton = MK_MBUTTON,
        XButton1 = MK_XBUTTON1,
        XButton2 = MK_XBUTTON2
};
// 255 keys, 7 mouseButtons
const size_t STATES_SIZE = 255 + 7;
struct Mouse
{
        UInt16 x;
```

```
		UInt16 y;
		UInt16 delta; // wheel delta
	};
	// [const][cop-c][cop-a][mov-c][mov-a]
	// [  +  ][  +  ][  +  ][  +  ][  +  ]
	struct GameInput
	{
		GameInput( )
		{
			std::fill_n( states.begin( ), STATES_SIZE, false );
		}
		GameInput( const GameInput& gi ) : states( gi.states ), mouse( gi.mouse ),
			text( gi.text )
		{ }
		GameInput( GameInput&& gi ) : states( std::move( gi.states ) ),
			mouse( std::move( gi.mouse ) ), text( std::move( gi.text ) )
		{ }
		GameInput operator = ( const GameInput& gi )
		{
			return GameInput{ gi };
		}
		GameInput operator = ( GameInput&& gi )
		{
			return GameInput{ std::move( gi ) };
		}
		bool& operator[]( size_t i )
		{
			return states[i];
		}
		bool& operator[]( Key k )
		{
			return states[static_cast<size_t>( k )];
		}
		bool& operator[]( MouseButton k )
		{
			return states[255 + static_cast<size_t>( k )];
		}
		bool operator[]( Key k ) const
		{
			return states[static_cast<size_t>( k )];
		}
		bool operator[]( MouseButton k ) const
		{
			return states[255 + static_cast<size_t>( k )];
		}
	private:
		GameInput( std::array<bool, STATES_SIZE> states, Mouse mouse, UInt32 text )
			: states( states ), mouse( mouse ), text( text )
		{ }
		std::array<bool, STATES_SIZE> states;
	public:
		Mouse mouse;
		UInt32 text;
	};
}

#pragma once
#include "gameInput.hpp"
#include "../utils/string.hpp"
namespace hp_ip
{
	enum struct WindowStyle : unsigned
	{
		Window,
		Fullscreen,
		Default = Window
	};
	struct WindowConfig
	{
		UInt width;
		UInt height;
		WindowStyle windowStyle;
		UInt bitsPerPx;
	};
```

```
        class Window
        {
        public:
                Window( const String& name, const WindowConfig& windowConfig ) : _handle(
nullptr ),
                        _name( s2ws( name ) ), _windowConfig( windowConfig ), _open( false )
                { }
                bool open( );
                void switchToFullscreen( );
                void processMessages( );
                bool isOpen( );
                void captureMouse( );
                void releaseMouse( );
                LRESULT CALLBACK windowProc( HWND handle, UINT message, WPARAM wParam, LPARAM
lParam );
                static LRESULT CALLBACK staticWindowProc( HWND handle, UINT message, WPARAM
wParam,
                        LPARAM lParam );
                static WindowConfig defaultWindowConfig( );
                static void setWindowVisibility( WindowHandle handle, const bool visible );
        private:
                bool isOnlyInstance( );
        private:
                WindowHandle _handle;
                std::wstring _name;
                WindowConfig _windowConfig;
                GameInput _gameInput;
                bool _open;
        public:
                WindowHandle handle( ) const
                {
                        return _handle;
                }
                GameInput gameInput( ) const
                {
                        return _gameInput;
                }
        };
}

#include <pch.hpp>
#include "../../include/core/timer.hpp"
#include "../../include/core/engine.hpp"
namespace hp_ip
{
        void Engine::run( const WindowConfig& windowConfig )
        {
                _pWindow = HP_NEW Window( _name, windowConfig );
                if ( _pWindow->open( ) )
                {
                        _pRenderer = HP_NEW Renderer( windowConfig );
                        if ( _pRenderer->init( _pWindow->handle( ) ) )
                        {
                                for ( auto& actor : _actors )
                                {
                                        actor.init( _resources, _pRenderer );
                                }
                                Timer timer;
                                while ( _pWindow->isOpen( ) )
                                {
                                        _pWindow->processMessages( );
                                        double deltaMs = timer.update( );
                                        for ( auto& actor : _actors )
                                        {
                                                actor.update( static_cast<float>( deltaMs ),
_pWindow->gameInput( ) );
                                        }
                                        _pRenderer->swapCameras( );
                                        _pRenderer->preRender( );
                                        for ( auto& actor : _actors )
                                        {
                                                actor.render( _pRenderer );
                                        }
                                        _pRenderer->present( );
```

```
                        }
                }
                else
                {
                        ERR( "Failed to initialize renderer." );
                }
        }
        else
        {
                ERR( "Failed to open a window." );
        }
}
void Engine::addActor( Actor&& actor )
{
        _actors.push_back( std::move( actor ) );
}
}

#include <pch.hpp>
#include "../../include/core/resources.hpp"
namespace hp_ip
{
        Resources::~Resources( )
        {
                for ( auto model : _loadedModels )
                {
                        HP_DELETE( model.second );
                }
                for ( auto model : _builtInModels )
                {
                        HP_DELETE( model.second );
                }
                for ( auto material : _materials )
                {
                        HP_DELETE( material.second );
                }
        };
        Model* Resources::getModel( Renderer* pRenderer, const LoadedModelDef& loadedModelDef
)
        {
                if ( _loadedModels.count( loadedModelDef ) == 0 )
                {
                        _loadedModels.emplace( loadedModelDef,
                                Model::loadModelFromFile( pRenderer, loadedModelDef ) );
                }
                return _loadedModels.at( loadedModelDef );
        }
        Model* Resources::getModel( Renderer* pRenderer, const BuiltInModelDef&
builtInModelDef )
        {
                if ( _builtInModels.count( builtInModelDef ) == 0 )
                {
                        switch ( std::get<0>( builtInModelDef ) )
                        {
                        case BuiltInModelType::Box:
                        {
                                _builtInModels.emplace( builtInModelDef,
                                        Model::cubeMesh( pRenderer, std::get<1>(
builtInModelDef ) ) );
                        }
                        break;
                        default:
                                WAR( "Missing built-In model for type " +
                                        std::to_string( static_cast<UInt8>( std::get<0>(
builtInModelDef ) ) ) + "." );
                        }
                }
                return _builtInModels.at( builtInModelDef );
        }
        Material* Resources::getMaterial( Renderer* pRenderer, const MaterialDef& materialDef
)
        {
                if ( _materials.count( materialDef ) == 0 )
                {
```

```
                        _materials.emplace( materialDef, Material::loadMaterial( pRenderer,
materialDef ) );
                }
                return _materials.at( materialDef );
        }
}

#include <pch.hpp>
#include "../../include/core/timer.hpp"
namespace hp_ip
{
        // http://randomascii.wordpress.com/2012/02/13/dont-store-that-in-a-float/
        // start timer at 2^32 to gain best precision
        const double TIME_ADDITION = 4294967296;
        double Timer::update( )
        {
                double currentTimeMs = getTimeMs( );
                double deltaMs = currentTimeMs - _lastTimeMs;
                _lastTimeMs = currentTimeMs;
                _timeMs += deltaMs;
                return deltaMs;
        }
        double Timer::getTimeMs( )
        {
                // use only single thread to calculate time
                HANDLE currentThread = GetCurrentThread( );
                DWORD_PTR previousMask = SetThreadAffinityMask( currentThread, 1 );
                static LARGE_INTEGER frequency;
                LARGE_INTEGER time;
                QueryPerformanceFrequency( &frequency );
                QueryPerformanceCounter( &time );
                // use previously used thread
                SetThreadAffinityMask( currentThread, previousMask );
                return double( 1000 * time.QuadPart / frequency.QuadPart );
        }
}

#include <pch.hpp>
#include "../../../include/core/actor/actor.hpp"
#include "../../../include/math/mat4x4.hpp"
namespace hp_ip
{
        void Actor::init( Resources& resources, Renderer* pRenderer )
        {
                for ( auto* component : _components )
                {
                        component->vInit( resources, pRenderer );
                }
                for ( auto& child : _children )
                {
                        child.init( resources, pRenderer );
                }
        }
        void Actor::update( const float deltaMs, const GameInput& input )
        {
                _pTransformComponent->vUpdate( deltaMs, input );
                Mat4x4 transform = _pTransformComponent->transform( );
                for ( auto* component : _components )
                {
                        component->vUpdate( deltaMs, input );
                }
                for ( auto& child : _children )
                {
                        child.transformComponent( ).setParentTransform( transform );
                        child.update( deltaMs, input );
                }
        }
        void Actor::render( Renderer* pRenderer )
        {
                for ( auto* component : _components )
                {
                        component->vRender( pRenderer );
                }
                for ( auto& child : _children )
```

```
                {
                        child.render( pRenderer );
                }
        }
        void Actor::addComponent( iComponent* component )
        {
                _components.push_back( component );
                component->_owner = this;
        }
        void Actor::addComponent( TransformComponent* component )
        {
                _pTransformComponent = component;
                _pTransformComponent->_owner = this;
        }
}

#include <pch.hpp>
#include "../../../../include/core/actor/component/cameraComponent.hpp"
#include "../../../../include/core/actor/actor.hpp"
#include "../../../../include/graphics/renderer.hpp"
namespace hp_ip
{
        void CameraComponent::vInit( Resources& resources, Renderer* pRenderer )
        {
                _pRenderer = pRenderer;
                _frustum = init( static_cast<float>( PI ) / 4.f,
                        static_cast<float>( pRenderer->windowConfig( ).width ) /
                        pRenderer->windowConfig( ).height,
                        _cameraDef.nearClipDist, _cameraDef.farClipDist );
                _projection = matrixPerspectiveFovLH( _frustum.fieldOfView,
                        _frustum.aspectRatio, _frustum.nearClipDist, _frustum.farClipDist );
        }
        void CameraComponent::vUpdate( const float deltaMs, const GameInput& input )
        {
                _pRenderer->setCamera( { _projection, _owner->transformComponent(
).modelTransform( ) } );
        }
}

#include <pch.hpp>
#include "../../../../include/core/actor/component/modelComponent.hpp"
#include "../../../../include/core/resources.hpp"
#include "../../../../include/core/actor/actor.hpp"
#include "../../../../include/graphics/renderer.hpp"
namespace hp_ip
{
        void ModelComponent::vInit( Resources& resources, Renderer* pRenderer )
        {
                switch ( _type )
                {
                case ModelType::Loaded:
                {
                        _model = resources.getModel( pRenderer, _loadedModelDef );
                }
                break;
                case ModelType::BuiltIn:
                {
                        _model = resources.getModel( pRenderer, _builtInModelDef );
                }
                break;
                default:
                        WAR( "Invalid model type." );
                }
                _material = resources.getMaterial( pRenderer, _materialDef );
        }
        void ModelComponent::vRender( Renderer* pRenderer )
        {
                if ( _owner != nullptr )
                {
                        const Camera& cam = pRenderer->getCamera( );
                        _material->setProjection( cam.projection );
                        _material->setView( inverse( cam.transform ) );
                        _material->setWorld( _owner->transformComponent( ).modelTransform( )
);
```

```
                        _material->setCameraPosition( pos( cam.transform ) );
                        _material->setAbientLightColor( Color( 0.1f, 0.1f, 0.1f, 0.6f ) );
                        _material->setDiffuseLightColor( Color( 1.0f, 0.95f, 0.4f, 0.4f ) );
                        _material->setSpecularLightColor( Color( 1.0f, 1.0f, 1.0f, 0.3f ) );
                        _material->setLightDirection( FVec3{ -0.5f, -1.0f, 0.1f } );
                        _material->setTextures( );
                        _material->setMaterials( );
                        _material->bindInputLayout( pRenderer );
                        for ( UInt32 i = 0; i < _material->getPassCount( ); ++i )
                        {
                                _material->applyPass( pRenderer, i );
                                for ( UInt32 j = 0; j < _model->_meshes.size( ); ++j )
                                {
                                        pRenderer->setBuffers( _model->_meshes[j] );
                                        pRenderer->drawIndexed( _model-
>_meshes[j].indicesSize( ), 0, 0 );
                                }
                        }
                }
        }
}

#include <pch.hpp>
#include "../../../../include/core/actor/component/transformComponent.hpp"
namespace hp_ip
{
        void TransformComponent::vUpdate( const float deltaMs, const GameInput& input )
        {
                _pos += _vel * deltaMs;
        }
        Mat4x4 TransformComponent::transform( ) const
        {
                return rotSclPosToMat4x4( _rot, _scl, _pos ) * _parentTransform;
        }
        Mat4x4 TransformComponent::modelTransform( ) const
        {
                return rotSclPosToMat4x4( _modelRot * _rot, _scl, _pos ) * _parentTransform;
        }
}

#include <pch.hpp>
#include "../../include/graphics/camera.hpp"
namespace hp_ip
{
        const Camera& CameraBuffer::getCamera( )
        {
                if ( _first )
                {
                        return _cam[0];
                }
                else
                {
                        return _cam[1];
                }
        }
        void CameraBuffer::setCamera( Camera&& camera )
        {
                if ( _first )
                {
                        _cam[1] = camera;
                }
                else
                {
                        _cam[0] = camera;
                }
        }
        void CameraBuffer::swap( )
        {
                _first = !_first;
        }
}

#include <pch.hpp>
#include "../../include/graphics/material.hpp"
```

```
#include "../../include/graphics/renderer.hpp"
#include "../../include/graphics/vertex.hpp"
namespace hp_ip
{
	Material* Material::loadMaterial( Renderer* pRenderer, const MaterialDef& materialDef
)
	{
		Material* material = HP_NEW Material
		{
			"assets/shaders/parallax.fx", // filename
			"Render", // techniqueName
			{ 1.0f, 1.0f }, // textureRepeat
			{ 0.5f, 0.5f, 0.5f }, // ambientMaterial
			{ 0.8f, 0.8f, 0.8f }, // diffuseMaterial
			{ 1.0f, 1.0f, 1.0f }, // specularMaterial
			25.0f // specularPower
		};
		bool materialLoaded;
		if ( materialLoaded = material->init( pRenderer ) )
		{
			if ( materialDef.diffuseTextureFilename != "" )
			{
				materialLoaded &= material->loadDiffuseTexture( pRenderer,
					materialDef.diffuseTextureFilename );
			}
			if ( materialDef.specularTextureFilename != "" )
			{
				materialLoaded &= material->loadSpecularTexture( pRenderer,
					materialDef.specularTextureFilename );
			}
			if ( materialDef.bumpTextureFilename != "" )
			{
				materialLoaded &= material->loadBumpTexture( pRenderer,
					materialDef.bumpTextureFilename );
			}
			if ( materialDef.parallaxTextureFilename != "" )
			{
				materialLoaded &= material->loadParallaxTexture( pRenderer,
					materialDef.parallaxTextureFilename );
			}
			if ( materialDef.textureRepeat.x != 0.0f &&
materialDef.textureRepeat.y != 0.0f )
			{
				material->_textureRepeat = materialDef.textureRepeat;
			}
		}
		if ( materialLoaded )
		{
			return material;
		}
		HP_DELETE( material );
		return nullptr;
	}
	Material::Material( const String& filename, const String& techniqueName,
		const FVec2& textureRepeat, const Color& ambientMaterial,
		const Color& diffuseMaterial, const Color& specularMaterial,
		const float specularPower ) : _filename( filename ), _techniqueName(
techniqueName ),
		_pEffect( nullptr ), _pTechnique( nullptr ), _pPass( nullptr ),
_pInputLayout( nullptr ),
		_techniqueDesc( ), _pViewMatrixVariable( nullptr ),
_pProjectionMatrixVariable( nullptr ),
		_pWorldMatrixVariable( nullptr ), _pDiffuseTextureVariable( nullptr ),
		_pSpecularTextureVariable( nullptr ), _pBumpTextureVariable( nullptr ),
		_pParallaxTextureVariable( nullptr ), _pEnvMapVariable( nullptr ),
		_pDiffuseTexture( nullptr ), _pSpecularTexture( nullptr ), _pBumpTexture(
nullptr ),
		_pParallaxTexture( nullptr ), _pEnvMapTexture( nullptr ), _textureRepeat(
textureRepeat ),
		_pUseDiffuseTextureVariable( nullptr ), _pUseSpecularTextureVariable( nullptr
),
		_pUseBumpTextureVariable( nullptr ), _pUseParallaxTextureVariable( nullptr ),
		_pAmbientLightColourVariable( nullptr ), _pDiffuseLightColourVariable(
nullptr ),
```

```
                _pSpecularLightColourVariable( nullptr ), _pLightDirectionVariable( nullptr
),
                _pAmbientMaterialVariable( nullptr ), _pDiffuseMaterialVariable( nullptr ),
                _pSpecularMaterialVariable( nullptr ), _pSpecularPowerVariable( nullptr ),
                _pTextureRepeatVariable( nullptr ), _pCameraPositionVariable( nullptr ),
                _ambientMaterial( ambientMaterial ), _diffuseMaterial( diffuseMaterial ),
                _specularMaterial( specularMaterial ), _specularPower( specularPower )
        {
                ZeroMemory( &_techniqueDesc, sizeof( D3D10_TECHNIQUE_DESC ) );
        }
        Material::~Material( )
        {
                HP_RELEASE( _pEnvMapTexture );
                HP_RELEASE( _pParallaxTexture );
                HP_RELEASE( _pBumpTexture );
                HP_RELEASE( _pSpecularTexture );
                HP_RELEASE( _pDiffuseTexture );
                HP_RELEASE( _pInputLayout );
                HP_RELEASE( _pEffect );
        }
        bool Material::init( Renderer* pRenderer )
        {
                if ( loadEffectFromFile( pRenderer ) )
                {
                        _pTechnique = _pEffect->GetTechniqueByName( _techniqueName.c_str( )
);
                        _pTechnique->GetDesc( &_techniqueDesc );
                        if ( createVertexLayout( pRenderer ) )
                        {
                                // retrieve all variables using semantic
                                _pWorldMatrixVariable = _pEffect->GetVariableBySemantic(
"WORLD" )->AsMatrix( );
                                _pViewMatrixVariable = _pEffect->GetVariableBySemantic(
"VIEW" )->AsMatrix( );
                                _pProjectionMatrixVariable = _pEffect-
>GetVariableBySemantic( "PROJECTION" )->AsMatrix( );
                                _pDiffuseTextureVariable = _pEffect->GetVariableByName(
"diffuseMap" )->AsShaderResource( );
                                _pSpecularTextureVariable = _pEffect->GetVariableByName(
"specularMap" )->AsShaderResource( );
                                _pBumpTextureVariable = _pEffect->GetVariableByName(
"bumpMap" )->AsShaderResource( );
                                _pParallaxTextureVariable = _pEffect->GetVariableByName(
"heightMap" )->AsShaderResource( );
                                _pEnvMapVariable = _pEffect->GetVariableByName( "envMap" )-
>AsShaderResource( );
                                // lights
                                _pAmbientLightColourVariable = _pEffect->GetVariableByName(
"ambientLightColour" )->AsVector( );
                                _pDiffuseLightColourVariable = _pEffect->GetVariableByName(
"diffuseLightColour" )->AsVector( );
                                _pSpecularLightColourVariable = _pEffect->GetVariableByName(
"specularLightColour" )->AsVector( );
                                _pLightDirectionVariable = _pEffect->GetVariableByName(
"lightDirection" )->AsVector( );
                                // materials
                                _pAmbientMaterialVariable = _pEffect->GetVariableByName(
"ambientMaterialColour" )->AsVector( );
                                _pDiffuseMaterialVariable = _pEffect->GetVariableByName(
"diffuseMaterialColour" )->AsVector( );
                                _pSpecularMaterialVariable = _pEffect->GetVariableByName(
"specularMaterialColour" )->AsVector( );
                                _pSpecularPowerVariable = _pEffect->GetVariableByName(
"specularPower" )->AsScalar( );
                                _pTextureRepeatVariable = _pEffect->GetVariableByName(
"textureRepeat" )->AsVector( );
                                // camera
                                _pCameraPositionVariable = _pEffect->GetVariableByName(
"cameraPosition" )->AsVector( );
                                // booleans
                                _pUseDiffuseTextureVariable = _pEffect->GetVariableByName(
"useDiffuseTexture" )->AsScalar( );
                                _pUseSpecularTextureVariable = _pEffect->GetVariableByName(
"useSpecularTexture" )->AsScalar( );
```

```
                                _pUseBumpTextureVariable = _pEffect->GetVariableByName(
"useBumpTexture" )->AsScalar( );
                                _pUseParallaxTextureVariable = _pEffect->GetVariableByName(
"useHeightTexture" )->AsScalar( );
                                return true;
                        }
                }
                return false;
        }
        bool Material::createVertexLayout( Renderer* pRenderer )
        {
                UInt32 elementsCount = ARRAYSIZE( D3D11_LAYOUT );
                D3DX11_PASS_DESC passDesc;
                _pTechnique->GetPassByIndex( 0 )->GetDesc( &passDesc );
                if ( FAILED( pRenderer->device( )->CreateInputLayout( D3D11_LAYOUT,
elementsCount,
                        passDesc.pIAInputSignature, passDesc.IAInputSignatureSize,
&_pInputLayout ) ) )
                {
                        return false;
                }
                return true;
        }
        bool Material::loadDiffuseTexture( Renderer* pRenderer, const String& filename )
        {
                return loadTexture( &_pDiffuseTexture, pRenderer, filename );
        }
        bool Material::loadSpecularTexture( Renderer* pRenderer, const String& filename )
        {
                return loadTexture( &_pSpecularTexture, pRenderer, filename );
        }
        bool Material::loadBumpTexture( Renderer* pRenderer, const String& filename )
        {
                return loadTexture( &_pBumpTexture, pRenderer, filename );
        }
        bool Material::loadParallaxTexture( Renderer* pRenderer, const String& filename )
        {
                return loadTexture( &_pParallaxTexture, pRenderer, filename );
        }
        bool Material::loadEnvMapTexture( Renderer* pRenderer, const String& filename )
        {
                return loadTexture( &_pEnvMapTexture, pRenderer, filename );
        }
        void Material::bindInputLayout( Renderer* pRenderer )
        {
                pRenderer->deviceContext( )->IASetInputLayout( _pInputLayout );
        }
        void Material::applyPass( Renderer* pRenderer, UInt32 i )
        {
                _pPass = _pTechnique->GetPassByIndex( i );
                _pPass->Apply( 0, pRenderer->deviceContext( ) );
        }
        bool Material::loadEffectFromFile( Renderer* pRenderer )
        {
                ID3DBlob* pBuffer = nullptr;
                if ( !compileD3DShader( _filename, "fx_5_0", &pBuffer ) )
                {
                        return false;
                }
                if ( FAILED( D3DX11CreateEffectFromMemory( pBuffer->GetBufferPointer( ),
                        pBuffer->GetBufferSize( ), 0, pRenderer->device( ), &_pEffect ) ) )
                {
                        HP_RELEASE( pBuffer );
                        return false;
                }
                return true;
        }
        bool Material::compileD3DShader( const String& filePath, const String& shaderModel,
                ID3DBlob** ppBuffer )
        {
                std::wstring wFilePath;
                wFilePath.assign( filePath.begin( ), filePath.end( ) );
                DWORD shaderFlags = D3DCOMPILE_ENABLE_STRICTNESS;
#       ifdef HP_DEBUG
```

```
                shaderFlags |= D3DCOMPILE_DEBUG;
#       endif
                ID3DBlob* errorBuffer = 0;
                HRESULT result;
                result = D3DCompileFromFile( wFilePath.c_str( ), 0, 0, 0, shaderModel.c_str(
),
                        shaderFlags, 0, ppBuffer, &errorBuffer );
                if ( FAILED( result ) )
                {
                        HP_RELEASE( errorBuffer );
                        ERR( "Error while compiling " + filePath + " shader." );
                        return false;
                }
                HP_RELEASE( errorBuffer );
                return true;
        }
        bool Material::loadTexture( ID3D11ShaderResourceView** texture, Renderer* pRenderer,
                const String& filename )
        {
                String fileExt = filename.substr( filename.find( '.' ) + 1 );
                std::transform( fileExt.begin( ), fileExt.end( ), fileExt.begin( ), ::tolower
);
                std::wstring wFilename = s2ws( filename );
                if ( fileExt.compare( "dds" ) == 0 )
                {
                        if ( FAILED( DirectX::CreateDDSTextureFromFile( pRenderer->device(
),
                                wFilename.c_str( ), nullptr, texture ) ) )
                        {
                                ERR( "Failed to load \"" + filename + "\" texture." );
                                return false;
                        }
                }
                else
                {
                        if ( FAILED( DirectX::CreateWICTextureFromFile( pRenderer->device(
),
                                wFilename.c_str( ), nullptr, texture ) ) )
                        {
                                ERR( "Failed to load \"" + filename + "\" texture." );
                                return false;
                        }
                }
                return true;
        }
}

#include <pch.hpp>
#include "../../include/graphics/model.hpp"
#include <algorithm>
#include <fbxsdk.h>
#pragma comment(lib, "libfbxsdk-md.lib")
#include "../../include/graphics/renderer.hpp"
namespace hp_ip
{
        Model::Model( )
        {

        }
        Model* Model::loadModelFromFile( Renderer* pRenderer, const LoadedModelDef&
loadedModelDef )
        {
                String filename = std::get<0>( loadedModelDef );
                String fileExt{ filename.substr( filename.find( '.' ) + 1 ) };
                std::transform( fileExt.begin( ), fileExt.end( ), fileExt.begin( ), ::tolower
);
                if ( fileExt.compare( "fbx" ) == 0 )
                {
                        return loadModelFromFBXFile( pRenderer, filename, std::get<1>(
loadedModelDef ) );
                }
                return nullptr;
        }
        Model* Model::loadModelFromFBXFile( Renderer* pRenderer, const String& filename,
```

```
                const float scale )
        {
                Model* model = HP_NEW Model( );
                FbxManager* manager = FbxManager::Create( );
                FbxIOSettings* settings = FbxIOSettings::Create( manager, IOSROOT );
                manager->SetIOSettings( settings );
                FbxImporter* importer = FbxImporter::Create( manager, "" );
                FbxGeometryConverter converter( manager );
                if ( !importer->Initialize( filename.c_str( ), -1, manager->GetIOSettings( )
) )
                {
                        HP_DELETE( model );
                        return nullptr;
                }
                FbxScene* scene = FbxScene::Create( manager, "myScene" );
                FbxAxisSystem axisSystem = scene->GetGlobalSettings( ).GetAxisSystem( );
                importer->Import( scene );
                importer->Destroy( );
                FbxNode* rootNode = scene->GetRootNode( );
                FbxMesh* fbxMesh = NULL;
                if ( rootNode )
                {
                        for ( Int32 i = 0; i < rootNode->GetChildCount( ); ++i )
                        {
                                FbxNode* childNode = rootNode->GetChild( i );
                                for ( Int32 j = 0; j < childNode->GetNodeAttributeCount( );
++j )
                                {
                                        FbxNodeAttribute* nodeAttribute = childNode-
>GetNodeAttributeByIndex( j );
                                        converter.Triangulate( nodeAttribute, true );
                                        if ( nodeAttribute->GetAttributeType( ) ==
FbxNodeAttribute::eMesh )
                                        {
                                                fbxMesh = (FbxMesh*) ( nodeAttribute );
                                                if ( fbxMesh )
                                                {
                                                        Mesh mesh;
                                                        FbxVector4* fbxVertices = fbxMesh-
>GetControlPoints( );
                                                        UInt32 vertexCount = fbxMesh-
>GetControlPointsCount( );
                                                        UInt32 indexCount = fbxMesh-
>GetPolygonVertexCount( );
                                                        Index* indices = (Index*) fbxMesh-
>GetPolygonVertices( );
                                                        Vertex* vertices = new
Vertex[vertexCount];
                                                        for ( UInt32 k = 0; k < vertexCount;
++k )
                                                        {
                                                                vertices[k].position.x =
float( fbxVertices[k][0] * scale );
                                                                vertices[k].position.y =
float( fbxVertices[k][1] * scale );
                                                                vertices[k].position.z =
float( fbxVertices[k][2] * scale );
                                                        }
                                                        for ( Int32 polyIndex = 0; polyIndex
< fbxMesh->GetPolygonCount( ); ++polyIndex )
                                                        {
                                                                for ( UInt32 vertexIndex =
0; vertexIndex < 3; ++vertexIndex )
                                                                {
                                                                        UInt32 cornerIndex
= fbxMesh->GetPolygonVertex( polyIndex, vertexIndex );
                                                                        FbxVector4
fbxVertex = fbxVertices[cornerIndex];
                                                                        FbxVector4
fbxNormal;
                                                                        fbxMesh-
>GetPolygonVertexNormal( polyIndex, vertexIndex, fbxNormal );
```

```
        fbxNormal.Normalize( );

        vertices[cornerIndex].normal = FVec3{ float( fbxNormal[0] ),
                                                                        float(
fbxNormal[1] ), float( fbxNormal[2] ) };
                                                                FbxVector2 fbxUV =
FbxVector2( 0, 0 );
                                                                FbxLayerElementUV*
fbxUVs = fbxMesh->GetLayer( 0 )->GetUVs( );
                                                                if ( fbxUVs )
                                                                {
                                                                        UInt32
UVIndex = 0;
                                                                        switch (
fbxUVs->GetMappingMode( ) )
                                                                        {
                                                                        case
FbxLayerElement::eByControlPoint:

        UVIndex = cornerIndex;

        break;
                                                                        case
FbxLayerElement::eByPolygonVertex:

        UVIndex = fbxMesh->GetTextureUVIndex( polyIndex, vertexIndex,
FbxLayerElement::eTextureDiffuse );

        break;
                                                                        }
                                                                        fbxUV =
fbxUVs->GetDirectArray( ).GetAt( UVIndex );

        vertices[cornerIndex].texCoord.x = float( fbxUV[0] );

        vertices[cornerIndex].texCoord.y = float( 1.0 - fbxUV[1] );
                                                                }
                                                        }
                                                }
                                                computeTangentsAndBinormals(
vertices, vertexCount, indices, indexCount );
                                                for ( UInt32 k = 0; k < vertexCount;
++k )
                                                {
                                                        addVertex( mesh,
vertices[k] );
                                                }
                                                for ( UInt32 k = 0; k < indexCount;
++k )
                                                {
                                                        addIndex( mesh, indices[k]
);
                                                }
                                                HP_DELETE_ARRAY( vertices );
                                                model->_meshes.push_back( std::move(
mesh ) );
                                        }
                                }
                        }
                }
                bool buffersCreated = true;
                for ( auto& mesh : model->_meshes )
                {
                        buffersCreated &= createBuffers( pRenderer, mesh );
                }
                if ( buffersCreated )
                {
                        settings->Destroy( );
                        manager->Destroy( );
                        return model;
                }
                else
                {
```

```
                                ERR( "Failed to initalize \"" + filename + "\"'s buffers."
);
                        }
                }
                ERR( "Failed to load \"" + filename + "\" model." );
                settings->Destroy( );
                manager->Destroy( );
                HP_DELETE( model );
                return nullptr;
        }
        Model* Model::cubeMesh( Renderer* pRenderer, const FVec3& dimensions )
        {
                Model* model = cubeMesh( dimensions );
                bool buffersCreated = true;
                for ( auto& mesh : model->_meshes )
                {
                        buffersCreated &= createBuffers( pRenderer, mesh );
                }
                if ( buffersCreated )
                {
                        return model;
                }
                ERR( "Failed to initalize cube's buffers." );
                HP_DELETE( model );
                return nullptr;
        }
        void Model::computeTangentsAndBinormals( Vertex* vertices, UInt32 vertexCount,
                UInt32* indices, UInt32 indexCount )
        {
                UInt32 triCount = indexCount / 3;
                FVec3* tangents = HP_NEW FVec3[vertexCount];
                FVec3* binormals = HP_NEW FVec3[vertexCount];
                for ( UInt32 i = 0; i < triCount; i += 3 )
                {
                        FVec3 v1 = vertices[indices[i]].position;
                        FVec3 v2 = vertices[indices[i + 1]].position;
                        FVec3 v3 = vertices[indices[i + 2]].position;
                        FVec2 uv1 = vertices[indices[i]].texCoord;
                        FVec2 uv2 = vertices[indices[i + 1]].texCoord;
                        FVec2 uv3 = vertices[indices[i + 2]].texCoord;
                        FVec3 edge1 = v2 - v1;
                        FVec3 edge2 = v3 - v1;
                        FVec2 edge1uv = uv2 - uv1;
                        FVec2 edge2uv = uv3 - uv1;
                        float cp = edge1uv.x * edge2uv.y - edge1uv.y * edge2uv.x;
                        if ( cp != 0.0f )
                        {
                                float mul = 1.0f / cp;
                                FVec3 tan = ( edge1 * edge2uv.y - edge2 * edge1uv.y ) * mul;
                                FVec3 binorm = ( edge1 * edge2uv.x - edge2 * edge1uv.x ) *
mul;
                                tangents[indices[i]] += tan;
                                binormals[indices[i]] += binorm;
                                tangents[indices[i + 1]] += tan;
                                binormals[indices[i + 1]] += binorm;
                                tangents[indices[i + 2]] += tan;
                                binormals[indices[i + 2]] += binorm;
                        }
                }
                for ( UInt32 i = 0; i < vertexCount; ++i )
                {
                        vertices[i].tangent = FVec3::normalize( tangents[i] );
                        vertices[i].binormal = FVec3::normalize( binormals[i] );
                }
                HP_DELETE_ARRAY( tangents );
                HP_DELETE_ARRAY( binormals );
        }
        void Model::addVertex( Mesh& mesh, const Vertex vertex )
        {
                mesh._vertices.push_back( vertex );
        }
        void Model::addIndex( Mesh& mesh, const Index index )
        {
                mesh._indices.push_back( index );
```

```
	}
	bool Model::createBuffers( Renderer* pRenderer, Mesh& mesh )
	{
		if ( pRenderer->createVertexBuffer( &mesh._vertexBuffer,
			sizeof( Vertex ) * mesh._vertices.size( ), &mesh._vertices.at( 0 ) )
)
		{
			if ( pRenderer->createIndexBuffer( &mesh._indexBuffer,
				sizeof( Index ) * mesh._indices.size( ), &mesh._indices.at(
0 ) ) )
			{
				return true;
			}
		}
		return false;
	}
	Model* Model::cubeMesh( const FVec3& dimensions )
	{
		Model* model = HP_NEW Model( );
		Mesh mesh;
		float halfWidth = dimensions.x / 2.f;
		float halfHeight = dimensions.y / 2.f;
		float halfLength = dimensions.z / 2.f;
		Vertex vertices[] =
		{
			{ FVec3{ -halfWidth, halfHeight, -halfLength }, Color( ), FVec2{
0.0f, 0.0f }, FVec3::up }, // 0 +Y (top face)
			{ FVec3{ halfWidth, halfHeight, -halfLength }, Color( ), FVec2{
1.0f, 0.0f }, FVec3::up }, // 1
			{ FVec3{ halfWidth, halfHeight, halfLength }, Color( ), FVec2{ 1.0f,
1.0f }, FVec3::up }, // 2
			{ FVec3{ -halfWidth, halfHeight, halfLength }, Color( ), FVec2{
0.0f, 1.0f }, FVec3::up }, // 3

			{ FVec3{ -halfWidth, -halfHeight, halfLength }, Color( ), FVec2{
0.0f, 0.0f }, -1 * FVec3::up }, // 4 -Y (bottom face)
			{ FVec3{ halfWidth, -halfHeight, halfLength }, Color( ), FVec2{
1.0f, 0.0f }, -1 * FVec3::up }, // 5
			{ FVec3{ halfWidth, -halfHeight, -halfLength }, Color( ), FVec2{
1.0f, 1.0f }, -1 * FVec3::up }, // 6
			{ FVec3{ -halfWidth, -halfHeight, -halfLength }, Color( ), FVec2{
0.0f, 1.0f }, -1 * FVec3::up }, // 7

			{ FVec3{ halfWidth, halfHeight, halfLength }, Color( ), FVec2{ 0.0f,
0.0f }, FVec3::right }, // 8 +X (right face)
			{ FVec3{ halfWidth, halfHeight, -halfLength }, Color( ), FVec2{
1.0f, 0.0f }, FVec3::right }, // 9
			{ FVec3{ halfWidth, -halfHeight, -halfLength }, Color( ), FVec2{
1.0f, 1.0f }, FVec3::right }, // 10
			{ FVec3{ halfWidth, -halfHeight, halfLength }, Color( ), FVec2{
0.0f, 1.0f }, FVec3::right }, // 11

			{ FVec3{ -halfWidth, halfHeight, -halfLength }, Color( ), FVec2{
0.0f, 0.0f }, -1 * FVec3::right }, // 12 -X (left face)
			{ FVec3{ -halfWidth, halfHeight, halfLength }, Color( ), FVec2{
1.0f, 0.0f }, -1 * FVec3::right }, // 13
			{ FVec3{ -halfWidth, -halfHeight, halfLength }, Color( ), FVec2{
1.0f, 1.0f }, -1 * FVec3::right }, // 14
			{ FVec3{ -halfWidth, -halfHeight, -halfLength }, Color( ), FVec2{
0.0f, 1.0f }, -1 * FVec3::right }, // 15

			{ FVec3{ -halfWidth, halfHeight, halfLength }, Color( ), FVec2{
0.0f, 0.0f }, FVec3::forward }, // 16 +Z (front face)
			{ FVec3{ halfWidth, halfHeight, halfLength }, Color( ), FVec2{ 1.0f,
0.0f }, FVec3::forward }, // 17
			{ FVec3{ halfWidth, -halfHeight, halfLength }, Color( ), FVec2{
1.0f, 1.0f }, FVec3::forward }, // 18
			{ FVec3{ -halfWidth, -halfHeight, halfLength }, Color( ), FVec2{
0.0f, 1.0f }, FVec3::forward }, // 19

			{ FVec3{ halfWidth, halfHeight, -halfLength }, Color( ), FVec2{
0.0f, 0.0f }, -1 * FVec3::forward }, // 20 -Z (back face)
			{ FVec3{ -halfWidth, halfHeight, -halfLength }, Color( ), FVec2{
1.0f, 0.0f }, -1 * FVec3::forward }, // 21
```

```
                        { FVec3{ -halfWidth, -halfHeight, -halfLength }, Color( ), FVec2{
1.0f, 1.0f }, -1 * FVec3::forward }, // 22
                        { FVec3{ halfWidth, -halfHeight, -halfLength }, Color( ), FVec2{
0.0f, 1.0f }, -1 * FVec3::forward }, // 23
                };
                Index indices[] =
                {
                        1, 0, 2, 2, 0, 3, // top face
                        5, 4, 6, 6, 4, 7, // bottom
                        9, 8, 10, 10, 8, 11, // right
                        13, 12, 14, 14, 12, 15, // left
                        17, 16, 18, 18, 16, 19, // front
                        21, 20, 22, 22, 20, 23 // back
                };
                computeTangentsAndBinormals( vertices, 24, indices, 36 );
                for ( UInt16 i = 0; i < 24; i++ )
                {
                        addVertex( mesh, vertices[i] );
                }
                for ( UInt16 i = 0; i < 36; i++ )
                {
                        addIndex( mesh, indices[i] );
                }
                model->_meshes.push_back( std::move( mesh ) );
                return model;
        }
}

#include <pch.hpp>
#include "../../include/graphics/renderer.hpp"
#include "../../include/graphics/mesh.hpp"
namespace hp_ip
{
        bool Renderer::init( WindowHandle windowHandle )
        {
                // driver types for fallback
                D3D_DRIVER_TYPE driverTypes[] = { D3D_DRIVER_TYPE_HARDWARE,
D3D_DRIVER_TYPE_WARP,
                        D3D_DRIVER_TYPE_SOFTWARE };
                UInt8 totalDriverTypes = ARRAYSIZE( driverTypes );
                // fallback feature levels
                D3D_FEATURE_LEVEL featureLevels[] = { D3D_FEATURE_LEVEL_11_0,
D3D_FEATURE_LEVEL_10_1,
                        D3D_FEATURE_LEVEL_10_0 };
                UInt8 totalFeatureLevels = ARRAYSIZE( featureLevels );
                // swap chain description
                DXGI_SWAP_CHAIN_DESC swapChainDesc;
                ZeroMemory( &swapChainDesc, sizeof( swapChainDesc ) );
                swapChainDesc.BufferCount = _windowConfig.windowStyle ==
WindowStyle::Fullscreen ? 2 : 1;
                swapChainDesc.BufferDesc.Width = _windowConfig.width;
                swapChainDesc.BufferDesc.Height = _windowConfig.height;
                swapChainDesc.BufferDesc.Format = DXGI_FORMAT_R8G8B8A8_UNORM;
                swapChainDesc.BufferDesc.RefreshRate.Numerator = 60;
                swapChainDesc.BufferDesc.RefreshRate.Denominator = 1;
                swapChainDesc.BufferUsage = DXGI_USAGE_RENDER_TARGET_OUTPUT;
                swapChainDesc.OutputWindow = windowHandle;
                swapChainDesc.Windowed = _windowConfig.windowStyle == WindowStyle::Window;
                swapChainDesc.SampleDesc.Count = 1;
                swapChainDesc.SampleDesc.Quality = 0;
                swapChainDesc.Flags = DXGI_SWAP_CHAIN_FLAG_ALLOW_MODE_SWITCH;
                // device creation flags
                UInt32 creationFlags = 0;
#       ifdef HP_DEBUG
                creationFlags |= D3D11_CREATE_DEVICE_DEBUG;
#       endif
                HRESULT result;
                UInt8 driver = 0;
                // loop through driver types and attempt to create device
                for ( driver; driver < totalDriverTypes; ++driver )
                {
                        result = D3D11CreateDeviceAndSwapChain( 0, driverTypes[driver], 0,
                                creationFlags, featureLevels, totalFeatureLevels,
D3D11_SDK_VERSION,
```

```
                        &swapChainDesc, &_pSwapChain, &_pDevice,
                        &_featureLevel, &_pDeviceContext );
                if ( SUCCEEDED( result ) )
                {
                        _driverType = driverTypes[driver];
                        break;
                }
        }
        if ( FAILED( result ) )
        {
                ERR( "Failed to create the Direct3D device!" );
                return false;
        }
        // back buffer texture to link render target with back buffer
        ID3D11Texture2D* backBufferTexture;
        if ( FAILED( _pSwapChain->GetBuffer( 0, _uuidof( ID3D11Texture2D ),
                (LPVOID*) &backBufferTexture ) ) )
        {
                ERR( "Failed to get the swap chain back buffer!" );
                return false;
        }
        D3D11_TEXTURE2D_DESC depthDesc;
        depthDesc.Width = _windowConfig.width;
        depthDesc.Height = _windowConfig.height;
        depthDesc.MipLevels = 1;
        depthDesc.ArraySize = 1;
        depthDesc.Format = DXGI_FORMAT_D32_FLOAT;
        depthDesc.SampleDesc.Count = 1;
        depthDesc.SampleDesc.Quality = 0;
        depthDesc.Usage = D3D11_USAGE_DEFAULT;
        depthDesc.BindFlags = D3D11_BIND_DEPTH_STENCIL;
        depthDesc.CPUAccessFlags = 0;
        depthDesc.MiscFlags = 0;
        if ( FAILED( _pDevice->CreateTexture2D( &depthDesc, NULL,
                &_pDepthStencilTexture ) ) )
        {
                ERR( "Failed to create depth stencil texture!" );
                return false;
        }
        D3D11_DEPTH_STENCIL_VIEW_DESC depthViewDesc;
        depthViewDesc.Format = depthDesc.Format;
        depthViewDesc.ViewDimension = D3D11_DSV_DIMENSION_TEXTURE2D;
        depthViewDesc.Texture2D.MipSlice = 0;
        depthViewDesc.Flags = 0;
        if ( FAILED( _pDevice->CreateDepthStencilView( _pDepthStencilTexture,
                &depthViewDesc, &_pDepthStencilView ) ) )
        {
                ERR( "Failed to create depth stencil view!" );
                return false;
        }
        if ( FAILED( _pDevice->CreateRenderTargetView( backBufferTexture, NULL,
                &_pRenderTargetView ) ) )
        {
                ERR( "Failed to create render target view!" );
                HP_RELEASE( backBufferTexture );
                return false;
        }
        HP_RELEASE( backBufferTexture );
        _pDeviceContext->OMSetRenderTargets( 1, &_pRenderTargetView,
                _pDepthStencilView );
        // setup the viewport
        D3D11_VIEWPORT viewport;
        viewport.Width = static_cast<FLOAT>( _windowConfig.width );
        viewport.Height = static_cast<FLOAT>( _windowConfig.height );
        viewport.MinDepth = 0.0f;
        viewport.MaxDepth = 1.0f;
        viewport.TopLeftX = 0.f;
        viewport.TopLeftY = 0.f;
        _pDeviceContext->RSSetViewports( 1, &viewport );
        _pDeviceContext->IASetPrimitiveTopology(
D3D11_PRIMITIVE_TOPOLOGY_TRIANGLELIST );
        return true;
}
void Renderer::preRender( )
```

```
        {
                float ClearColor[4] = { 0.0f, 0.125f, 0.3f, 1.0f };
                _pDeviceContext->ClearRenderTargetView( _pRenderTargetView,
                        ClearColor );
                _pDeviceContext->ClearDepthStencilView( _pDepthStencilView,
                        D3D11_CLEAR_DEPTH, 1.0f, 0 );
        }
        void Renderer::swapCameras( )
        {
                _cameraBuffer.swap( );
        }
        void Renderer::present( )
        {
                _pSwapChain->Present( 0, 0 );
        }
        bool Renderer::createVertexBuffer( ID3D11Buffer** vertexBuffer, UInt32 byteWidth,
                const Vertex* initData )
        {
                D3D11_BUFFER_DESC bd;
                bd.Usage = D3D11_USAGE_DEFAULT;
                bd.ByteWidth = byteWidth;
                bd.BindFlags = D3D11_BIND_VERTEX_BUFFER;
                bd.CPUAccessFlags = 0;
                bd.MiscFlags = 0;
                D3D11_SUBRESOURCE_DATA subresourceData;
                subresourceData.pSysMem = initData;
                subresourceData.SysMemPitch = 0;
                subresourceData.SysMemSlicePitch = 0;
                if ( SUCCEEDED( _pDevice->CreateBuffer( &bd, &subresourceData, vertexBuffer )
) )
                {
                        return true;
                }
                return false;
        }
        bool Renderer::createIndexBuffer( ID3D11Buffer** indexBuffer, UInt32 byteWidth,
                const Index* initData )
        {
                D3D11_BUFFER_DESC bd;
                bd.Usage = D3D11_USAGE_DEFAULT;
                bd.ByteWidth = byteWidth;
                bd.BindFlags = D3D11_BIND_INDEX_BUFFER;
                bd.CPUAccessFlags = 0;
                bd.MiscFlags = 0;
                D3D11_SUBRESOURCE_DATA subresourceData;
                subresourceData.pSysMem = initData;
                subresourceData.SysMemPitch = 0;
                subresourceData.SysMemSlicePitch = 0;
                if ( SUCCEEDED( _pDevice->CreateBuffer( &bd, &subresourceData, indexBuffer )
) )
                {
                        return true;
                }
                return false;
        }
        void Renderer::setVertexBuffers( ID3D11Buffer** vertexBuffer, UInt32* stride,
                UInt32* offset )
        {
                _pDeviceContext->IASetVertexBuffers( 0, 1, vertexBuffer, stride, offset );
        }
        void Renderer::setIndexBuffer( ID3D11Buffer** indexBuffer )
        {
                _pDeviceContext->IASetIndexBuffer( *indexBuffer, DXGI_FORMAT_R32_UINT, 0 );
        }
        void Renderer::setBuffers( Mesh& mesh )
        {
                UInt32 stride = sizeof( Vertex );
                UInt32 offset = 0;
                setVertexBuffers( &mesh._vertexBuffer, &stride, &offset );
                setIndexBuffer( &mesh._indexBuffer );
        }
        void Renderer::drawIndexed( UInt32 indexCount, UInt32 startIndexLocation,
                UInt32 baseVertexLocation )
        {
```

```
                _pDeviceContext->DrawIndexed( indexCount, startIndexLocation,
baseVertexLocation );
        }
}

///
/// replace WinMain function with main on Windows
///
#include "pch.hpp"

#ifdef HP_PLATFORM_WIN32
extern int main( int argc, char ** argv );

int WINAPI WinMain( HINSTANCE, HINSTANCE, LPSTR, INT )
{
        return main( __argc, __argv );
}
#endif

#include <pch.hpp>
#include "../../include/math/frustum.hpp"
namespace hp_ip
{
        Frustum init( const float fieldOfView, const float aspectRatio, const float
nearClipDist,
                const float farClipDist )
        {
                Frustum frustum;
                frustum.fieldOfView = fieldOfView;
                frustum.aspectRatio = aspectRatio;
                frustum.nearClipDist = nearClipDist;
                frustum.farClipDist = farClipDist;
                float tanHalfFov = tan( frustum.fieldOfView / 2.f );
                FVec3 nearRight = ( frustum.nearClipDist * tanHalfFov ) * frustum.aspectRatio
* FVec3::right;
                FVec3 farRight = ( frustum.farClipDist * tanHalfFov ) * frustum.aspectRatio *
FVec3::right;
                FVec3 nearUp = ( frustum.nearClipDist * tanHalfFov ) * frustum.aspectRatio *
FVec3::up;
                FVec3 farUp = ( frustum.farClipDist * tanHalfFov ) * frustum.aspectRatio *
FVec3::up;
                frustum.nearClipVerts[0] = ( frustum.nearClipDist * FVec3::forward ) -
nearRight + nearUp;
                frustum.nearClipVerts[1] = ( frustum.nearClipDist * FVec3::forward ) +
nearRight + nearUp;
                frustum.nearClipVerts[2] = ( frustum.nearClipDist * FVec3::forward ) +
nearRight - nearUp;
                frustum.nearClipVerts[3] = ( frustum.nearClipDist * FVec3::forward ) -
nearRight - nearUp;
                frustum.farClipVerts[0] = ( frustum.farClipDist * FVec3::forward ) - farRight
+ farUp;
                frustum.farClipVerts[1] = ( frustum.farClipDist * FVec3::forward ) + farRight
+ farUp;
                frustum.farClipVerts[2] = ( frustum.farClipDist * FVec3::forward ) + farRight
- farUp;
                frustum.farClipVerts[3] = ( frustum.farClipDist * FVec3::forward ) - farRight
- farUp;
                FVec3 origin;
                frustum.planes[static_cast<UInt8>( FrustumSides::Near )] = init(
frustum.nearClipVerts[2], frustum.nearClipVerts[1], frustum.nearClipVerts[0] );
                frustum.planes[static_cast<UInt8>( FrustumSides::Far )] = init(
frustum.farClipVerts[0], frustum.farClipVerts[1], frustum.farClipVerts[2] );
                frustum.planes[static_cast<UInt8>( FrustumSides::Right )] = init(
frustum.farClipVerts[2], frustum.farClipVerts[1], origin );
                frustum.planes[static_cast<UInt8>( FrustumSides::Top )] = init(
frustum.farClipVerts[1], frustum.farClipVerts[0], origin );
                frustum.planes[static_cast<UInt8>( FrustumSides::Left )] = init(
frustum.farClipVerts[0], frustum.farClipVerts[3], origin );
                frustum.planes[static_cast<UInt8>( FrustumSides::Bottom )] = init(
frustum.farClipVerts[3], frustum.farClipVerts[2], origin );
                return frustum;
        }
        bool isInside( const Frustum& frustum, const FVec3& point, const float radius )
        {
```

```
                for ( UInt8 i = 0; i < static_cast<UInt8>( FrustumSides::Count ); ++i )
                {
                        if ( !isInside( frustum.planes[i], point, radius ) )
                        {
                                return false;
                        }
                }
                return true;
        }
}

#include <pch.hpp>
#include "../../include/math/mat4x4.hpp"
namespace hp_ip
{
        const Mat4x4 Mat4x4::identity( Mat4x4( 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1
) );
        FVec3 pos( const Mat4x4& mat )
        {
                return FVec3{ mat.m[3][0], mat.m[3][1], mat.m[3][2] };
        }
        float determinant( const Mat4x4& mat )
        {
                FVec4 v1, v2, v3;
                v1.x = mat.m[0][0];
                v1.y = mat.m[1][0];
                v1.z = mat.m[2][0];
                v1.w = mat.m[3][0];
                v2.x = mat.m[0][1];
                v2.y = mat.m[1][1];
                v2.z = mat.m[2][1];
                v2.w = mat.m[3][1];
                v3.x = mat.m[0][2];
                v3.y = mat.m[1][2];
                v3.z = mat.m[2][2];
                v3.w = mat.m[3][2];
                FVec4 x = cross( v1, v2, v3 );
                return -( mat.m[0][3] * x.x + mat.m[1][3] * x.y + mat.m[2][3] * x.z +
                        mat.m[3][3] * x.w );
        }
        Mat4x4 inverse( const Mat4x4& mat )
        {
                Mat4x4 invMat;
                FVec4 vec[3];
                int i, j;
                float det = determinant( mat );
                for ( i = 0; i < 4; ++i )
                {
                        for ( j = 0; j < 4; ++j )
                        {
                                if ( i != j )
                                {
                                        int a = ( j <= i ) ? j : j - 1;
                                        vec[a].x = mat.m[j][0];
                                        vec[a].y = mat.m[j][1];
                                        vec[a].z = mat.m[j][2];
                                        vec[a].w = mat.m[j][3];
                                }
                        }
                        FVec4 x = cross( vec[0], vec[1], vec[2] );
                        float cofactor;
                        for ( j = 0; j < 4; ++j )
                        {
                                switch ( j )
                                {
                                case 0: cofactor = x.x; break;
                                case 1: cofactor = x.y; break;
                                case 2: cofactor = x.z; break;
                                case 3: cofactor = x.w; break;
                                }
                                invMat.m[j][i] = pow( -1.0f, i ) * cofactor / det;
                        }
                }
                return invMat;
```

```
        }
        Mat4x4 matrixPerspectiveFovLH( const float fieldOfView, const float aspectRatio,
                const float nearClipDist, const float farClipDist )
        {
                Mat4x4 perspective;
                float tanFov = tan( fieldOfView / 2.0f );
                perspective.m[0][0] = 1.0f / ( aspectRatio * tanFov );
                perspective.m[1][1] = 1.0f / tanFov;
                perspective.m[2][2] = farClipDist / ( farClipDist - nearClipDist );
                perspective.m[2][3] = 1.0f;
                perspective.m[3][2] = ( farClipDist *  -nearClipDist ) / ( farClipDist -
nearClipDist );
                perspective.m[3][3] = 0.0f;
                return perspective;
        }
        Mat4x4 rotToMat4x4( const FQuat& rot )
        {
                Mat4x4 mat;
                mat.m[0][0] = 1.0f - 2.0f * ( rot.y * rot.y + rot.z * rot.z );
                mat.m[0][1] = 2.0f * ( rot.x *rot.y + rot.z * rot.w );
                mat.m[0][2] = 2.0f * ( rot.x * rot.z - rot.y * rot.w );
                mat.m[1][0] = 2.0f * ( rot.x * rot.y - rot.z * rot.w );
                mat.m[1][1] = 1.0f - 2.0f * ( rot.x * rot.x + rot.z * rot.z );
                mat.m[1][2] = 2.0f * ( rot.y *rot.z + rot.x *rot.w );
                mat.m[2][0] = 2.0f * ( rot.x * rot.z + rot.y * rot.w );
                mat.m[2][1] = 2.0f * ( rot.y *rot.z - rot.x *rot.w );
                mat.m[2][2] = 1.0f - 2.0f * ( rot.x * rot.x + rot.y * rot.y );
                return mat;
        }
        Mat4x4 posToMat4x4( const FVec3& pos )
        {
                Mat4x4 mat;
                mat.m[3][0] = pos.x;
                mat.m[3][1] = pos.y;
                mat.m[3][2] = pos.z;
                return mat;
        }
        Mat4x4 sclToMat4x4( const FVec3& scl )
        {
                Mat4x4 mat;
                mat.m[0][0] = scl.x;
                mat.m[1][1] = scl.y;
                mat.m[2][2] = scl.z;
                return mat;
        }
        Mat4x4 rotSclPosToMat4x4( const FQuat& rot, const FVec3& scl, const FVec3& pos )
        {
                Mat4x4 mat;
                mat.m[0][0] = 1.0f - 2.0f * ( rot.y * rot.y + rot.z * rot.z );
                mat.m[0][1] = 2.0f * ( rot.x *rot.y + rot.z * rot.w );
                mat.m[0][2] = 2.0f * ( rot.x * rot.z - rot.y * rot.w );
                mat.m[1][0] = 2.0f * ( rot.x * rot.y - rot.z * rot.w );
                mat.m[1][1] = 1.0f - 2.0f * ( rot.x * rot.x + rot.z * rot.z );
                mat.m[1][2] = 2.0f * ( rot.y *rot.z + rot.x *rot.w );
                mat.m[2][0] = 2.0f * ( rot.x * rot.z + rot.y * rot.w );
                mat.m[2][1] = 2.0f * ( rot.y *rot.z - rot.x *rot.w );
                mat.m[2][2] = 1.0f - 2.0f * ( rot.x * rot.x + rot.y * rot.y );
                mat.m[3][0] = pos.x;
                mat.m[3][1] = pos.y;
                mat.m[3][2] = pos.z;
                mat.m[0][0] *= scl.x;
                mat.m[1][1] *= scl.y;
                mat.m[2][2] *= scl.z;
                return mat;

        }
}

#include <pch.hpp>
#include "../../include/math/plane.hpp"
namespace hp_ip
{
        Plane init( const FVec3& p0, const FVec3& p1, const FVec3& p2 )
        {
```

```
                FVec3 edge1, edge2, normal;
                edge1 = p1 - p0;
                edge2 = p2 - p0;
                normal = FVec3::normalize( cross( edge1, edge2 ) );
                return normalize( planeFromPointNormal( p0, normal ) );
        }
        Plane normalize( const Plane& plane )
        {
                Plane normPlane;
                float mag;
                mag = 1.0f / sqrt( plane.a * plane.a + plane.b * plane.b + plane.c * plane.c
);
                normPlane.a = plane.a * mag;
                normPlane.b = plane.b * mag;
                normPlane.c = plane.c * mag;
                normPlane.d = plane.d * mag;
                return normPlane;
        }
        Plane planeFromPointNormal( const FVec3& point, const FVec3& normal )
        {
                Plane plane;
                plane.a = normal.x;
                plane.b = normal.y;
                plane.c = normal.z;
                plane.d = dot( point, normal );
                return plane;
        }
        bool isInside( const Plane& plane, const FVec3& point, const float radius )
        {
                float distance;
                distance = planeDotCoord( plane, point );
                return distance >= -radius;
        }
        float planeDotCoord( const Plane& plane, const FVec3& point )
        {
                return plane.a * point.x + plane.b * point.y + plane.c * point.z + plane.d;
        }
}

#include <pch.hpp>
#include "../../include/math/quat.hpp"
namespace hp_ip
{
        const FQuat FQuat::identity( FQuat( 0.0f, 0.0f, 0.0f, 1.0f ) );
}

#include <pch.hpp>
#include "../../include/math/vec3.hpp"
namespace hp_ip
{
        const FVec3 FVec3::zero{ 0.f, 0.f, 0.f };
        const FVec3 FVec3::right{ 1.f, 0.f, 0.f };
        const FVec3 FVec3::up{ 0.f, 1.f, 0.f };
        const FVec3 FVec3::forward{ 0.f, 0.f, 1.f };
}

#include <pch.hpp>
#include "../../include/math/vec4.hpp"
namespace hp_ip
{
        template<typename A>
        Vec4<A>::Vec4( const FVec3& vec ) : x( vec.x ), y( vec.y ), z( vec.z ), w( 1.f )
        { }
        const FVec4 FVec4::right( 1.f, 0.f, 0.f, 0.f );
        const FVec4 FVec4::up( 0.f, 1.f, 0.f, 0.f );
        const FVec4 FVec4::forward( 0.f, 0.f, 1.f, 0.f );
}

#include <pch.hpp>
#include "../../include/utils/string.hpp"
namespace hp_ip
{
        std::wstring s2ws( const std::string& s )
        {
```

```
                int len;
                int slength = (int) s.length( ) + 1;
                len = MultiByteToWideChar( CP_ACP, 0, s.c_str( ), slength, 0, 0 );
                wchar_t* buf = new wchar_t[len];
                MultiByteToWideChar( CP_ACP, 0, s.c_str( ), slength, buf, len );
                std::wstring r( buf );
                delete[] buf;
                return r;
        }
        std::string basePath( const std::string& path )
        {
                size_t pos = path.find_last_of( "\\/" );
                return ( std::string::npos == pos ) ? "" : path.substr( 0, pos + 1 );
        }
}

#include <pch.hpp>
#include "../../include/window/window.hpp"
namespace hp_ip
{
        bool Window::open( )
        {
                if ( isOnlyInstance( ) )
                {
                        // window class details
                        WNDCLASSEX windowClass = { 0 };
                        windowClass.cbSize = sizeof( WNDCLASSEX );
                        windowClass.style = CS_VREDRAW | CS_HREDRAW;
                        windowClass.lpfnWndProc = &Window::staticWindowProc;
                        windowClass.cbClsExtra = 0;
                        windowClass.cbWndExtra = 0;
                        windowClass.hInstance = GetModuleHandle( nullptr );
                        windowClass.hIcon = 0;
                        windowClass.hIconSm = 0;
                        windowClass.hCursor = 0;
                        windowClass.hbrBackground = 0;
                        windowClass.lpszMenuName = 0;
                        windowClass.lpszClassName = _name.c_str( );
                        // register the window
                        if ( RegisterClassEx( &windowClass ) )
                        {
                                // find position and size
                                HDC screenDC = GetDC( nullptr );
                                unsigned left = ( GetDeviceCaps( screenDC, HORZRES ) -
_windowConfig.width ) / 2;
                                unsigned top = ( GetDeviceCaps( screenDC, VERTRES ) -
_windowConfig.height ) / 2;
                                unsigned width = _windowConfig.width;
                                unsigned height = _windowConfig.height;
                                ReleaseDC( nullptr, screenDC );
                                // set the style of the window
                                DWORD style = WS_VISIBLE;
                                if ( _windowConfig.windowStyle == WindowStyle::Window )
                                {
                                        style |= WS_CAPTION | WS_MINIMIZEBOX | WS_THICKFRAME
| WS_MAXIMIZEBOX | WS_SYSMENU;
                                        // adjust the window size with the borders etc.
                                        RECT rectangle = { 0, 0, _windowConfig.width,
_windowConfig.height };
                                        AdjustWindowRect( &rectangle, style, false );
                                        width = rectangle.right - rectangle.left;
                                        height = rectangle.bottom - rectangle.top;
                                }
                                // create the window
                                _handle = CreateWindowEx( 0, _name.c_str( ), _name.c_str( ),
style, left, top, width, height,
                                        GetDesktopWindow( ), nullptr, GetModuleHandle(
nullptr ), this );
                                if ( _handle == nullptr )
                                {
                                        ERR( GetLastError( ) );
                                        return false;
                                }
```

```
                                else if ( _windowConfig.windowStyle ==
WindowStyle::Fullscreen )
                                {
                                        switchToFullscreen( );
                                }
                                _open = true;
                        }
                        else
                        {
                                ERR( GetLastError( ) );
                        }
                }
                return _open;
        }
        void Window::switchToFullscreen( )
        {
                // set display settings
                DEVMODE devMode;
                devMode.dmSize = sizeof( devMode );
                devMode.dmPelsWidth = _windowConfig.width;
                devMode.dmPelsHeight = _windowConfig.height;
                devMode.dmBitsPerPel = _windowConfig.bitsPerPx;
                devMode.dmFields = DM_PELSWIDTH | DM_PELSHEIGHT | DM_BITSPERPEL;
                // change default display device settings
                if ( ChangeDisplaySettings( &devMode, CDS_FULLSCREEN ) !=
DISP_CHANGE_SUCCESSFUL )
                {
                        return;
                }
                // set window style
                SetWindowLong( _handle, GWL_STYLE, WS_POPUP | WS_CLIPCHILDREN |
WS_CLIPSIBLINGS );
                // set extended window style
                SetWindowLong( _handle, GWL_EXSTYLE, WS_EX_APPWINDOW );
                // set window size, position and z-order
                SetWindowPos( _handle, HWND_TOP, 0, 0, _windowConfig.width,
_windowConfig.height,
                        SWP_FRAMECHANGED );
                // show the window
                setWindowVisibility( _handle, true );
        }
        void Window::processMessages( )
        {
                MSG message;
                while ( PeekMessage( &message, _handle, 0, 0, PM_REMOVE ) )
                {
                        TranslateMessage( &message );
                        DispatchMessage( &message );
                }
        }
        bool Window::isOpen( )
        {
                return _open;
        }
        void Window::captureMouse( )
        {
                SetCapture( _handle );
        }
        void Window::releaseMouse( )
        {
                ReleaseCapture( );
        }
        LRESULT CALLBACK Window::windowProc( HWND handle, UINT message, WPARAM wParam,
                LPARAM lParam )
        {
                switch ( message )
                {
                case WM_CLOSE:
                {
                        _open = false;
                        break;
                }
                case WM_ACTIVATEAPP:
                {
```

```
                        break;
                }
                case WM_KEYDOWN:
                case WM_SYSKEYDOWN:
                {
                        _gameInput[static_cast<Key>( wParam )] = true;
                        break;
                }
                case WM_KEYUP:
                case WM_SYSKEYUP:
                {
                        _gameInput[static_cast<Key>( wParam )] = false;
                        break;
                }
                case WM_CHAR:
                {
                        _gameInput.text = wParam;
                        break;
                }
                case WM_MOUSEMOVE:
                {
                        _gameInput.mouse.x = LOWORD( lParam );
                        _gameInput.mouse.y = HIWORD( lParam );
                        break;
                }
                case WM_LBUTTONDOWN:
                {
                        captureMouse( );
                        _gameInput[MouseButton::LeftButton] = true;
                        break;
                }
                case WM_LBUTTONUP:
                {
                        releaseMouse( );
                        _gameInput[MouseButton::LeftButton] = false;
                        break;
                }
                case WM_RBUTTONDOWN:
                {
                        captureMouse( );
                        _gameInput[MouseButton::RightButton] = true;
                        break;
                }
                case WM_RBUTTONUP:
                {
                        releaseMouse( );
                        _gameInput[MouseButton::RightButton] = false;
                        break;
                }
                case WM_MBUTTONDOWN:
                {
                        captureMouse( );
                        _gameInput[MouseButton::MiddleButton] = true;
                        break;
                }
                case WM_MBUTTONUP:
                {
                        releaseMouse( );
                        _gameInput[MouseButton::MiddleButton] = false;
                        break;
                }
                case WM_MOUSEWHEEL:
                {
                        _gameInput.mouse.delta = HIWORD( wParam ) / 120;
                        break;
                }
                case WM_XBUTTONDOWN:
                {
                        captureMouse( );
                        _gameInput[HIWORD( wParam ) == XBUTTON1 ? MouseButton::XButton1 :
MouseButton::XButton2] = true;
                        break;
                }
                case WM_XBUTTONUP:
```

```
                {
                        releaseMouse( );
                        _gameInput[HIWORD( wParam ) == XBUTTON1 ? MouseButton::XButton1 :
MouseButton::XButton2] = false;
                        break;
                }
                default:
                        return DefWindowProc( handle, message, wParam, lParam );
                }
                return FALSE;
        }
        LRESULT CALLBACK Window::staticWindowProc( HWND handle, UINT message, WPARAM wParam,
                LPARAM lParam )
        {
                Window* window = reinterpret_cast<Window*>( GetWindowLongPtr( handle,
GWL_USERDATA ) );
                switch ( message )
                {
                case WM_CREATE:
                {
                        CREATESTRUCT *cs = reinterpret_cast<CREATESTRUCT*>( lParam );
                        window = reinterpret_cast<Window*>( cs->lpCreateParams );
                        SetLastError( 0 );
                        if ( SetWindowLongPtr( handle, GWL_USERDATA,
reinterpret_cast<LONG_PTR>( window ) )
                                == 0 )
                        {
                                if ( GetLastError( ) != 0 )
                                {
                                        ERR( "Unable to set window user data." );
                                        return FALSE;
                                }
                        }
                        break;
                }
                default:
                        return window->windowProc( handle, message, wParam, lParam );
                }
                return FALSE;
        }
        WindowConfig Window::defaultWindowConfig( )
        {
                DEVMODE mode;
                mode.dmSize = sizeof( mode );
                EnumDisplaySettings( nullptr, ENUM_CURRENT_SETTINGS, &mode );
                return WindowConfig{ mode.dmPelsWidth, mode.dmPelsHeight,
WindowStyle::Default, mode.dmBitsPerPel };
        }
        void Window::setWindowVisibility( WindowHandle handle, const bool visible )
        {
                ShowWindow( handle, visible ? SW_SHOW : SW_HIDE );
                if ( visible )
                {
                        SetFocus( handle );
                        SetForegroundWindow( handle );
                        SetActiveWindow( handle );
                }
        }
        bool Window::isOnlyInstance( )
        {
                HANDLE handle = CreateMutex( nullptr, true, _name.c_str( ) );
                if ( GetLastError( ) != ERROR_SUCCESS )
                {
                        WindowHandle windowHandle = FindWindow( _name.c_str( ), nullptr );
                        if ( windowHandle != nullptr )
                        {
                                Window::setWindowVisibility( windowHandle, true );
                                return false;
                        }
                }
                return true;
        }
}
```