3D 引擎开发笔记

Zhang Yufeng [1]

2022 年 5 月 14 日

[1]Email: 759094438@qq.com

目录

第一部分

序言

序言

0.1 新的开始

从本科时期开始接触代码后，我便一直希望能够进行自己的项目开发，日常也喜欢写一些个人小程序自娱自乐，因为一直对游戏较为感兴趣，因此初期希望研究并开发类似 Unity 的游戏引擎。当然 Unity 是一个庞然大物，仅靠一个人是无法完成开发的，但是把目标放到 Unity 的部分功能来，这个计划就可行的多。困于本科时期知识基础的薄弱，以及相关开发经验的缺乏，四年间都只做过一些较小的个人项目，我的知识尚未形成一个较为完整的体系，也没有将个人的思想融入到项目中实现。

研究生阶段，为了在读研生涯中做点什么，希望留下自己的印记，我开始了这个项目的设计。但是由于毕业与工作的压力，之前的文章写了部分就暂时搁浅了，而现在终于有时间来继续推进。我的目标是将此项目作为一个小的完整的系统，搜集资料后按照个人的想法进行设计和实现，不仅能够锻炼提升个人水平，还能够锻炼阅读写作与其他能力。同时，这个系列的文章不仅是我个人学习过程的笔记，也将作为后续编写类似程序的相关参考，并在这之中提出一些我个人的想法来抛砖引玉，得到他人的指导或同好的学习与分享。最后希望通过此项目对其他做类似学习与开发的人做出一些微薄的贡献。

在这之中，感谢我的亲人、我的对象、我的老师与朋友对我的支持与帮助。

本文使用 LaTeX [1] 编写。使用 XeLaTeX 编译。

0.2 初步的写作计划

此项目聚焦于跨平台 3D 引擎的设计开发，从零开始搭建一个 3D 引擎用于渲染 3D 场景，以及支持脚本及 AI 的场景内物体互动，使其具有成为仿真环境的可能性。本项目模仿游戏引擎进行模块划分，一个游戏引擎由许多模块组成，每个模块具有相对独立的功

[1] https://mirrors.tuna.tsinghua.edu.cn/CTAN/info/lshort/chinese/lshort-zh-cn.pdf

能，参考《游戏引擎架构》的内容，本文章将按照以下几个部分设计介绍。当然也有部分章节可能脱离主线，介绍一些额外的内容。

本笔记主要参考的内容出自大名鼎鼎的 Game Engine Architecture[2]的引擎架构：

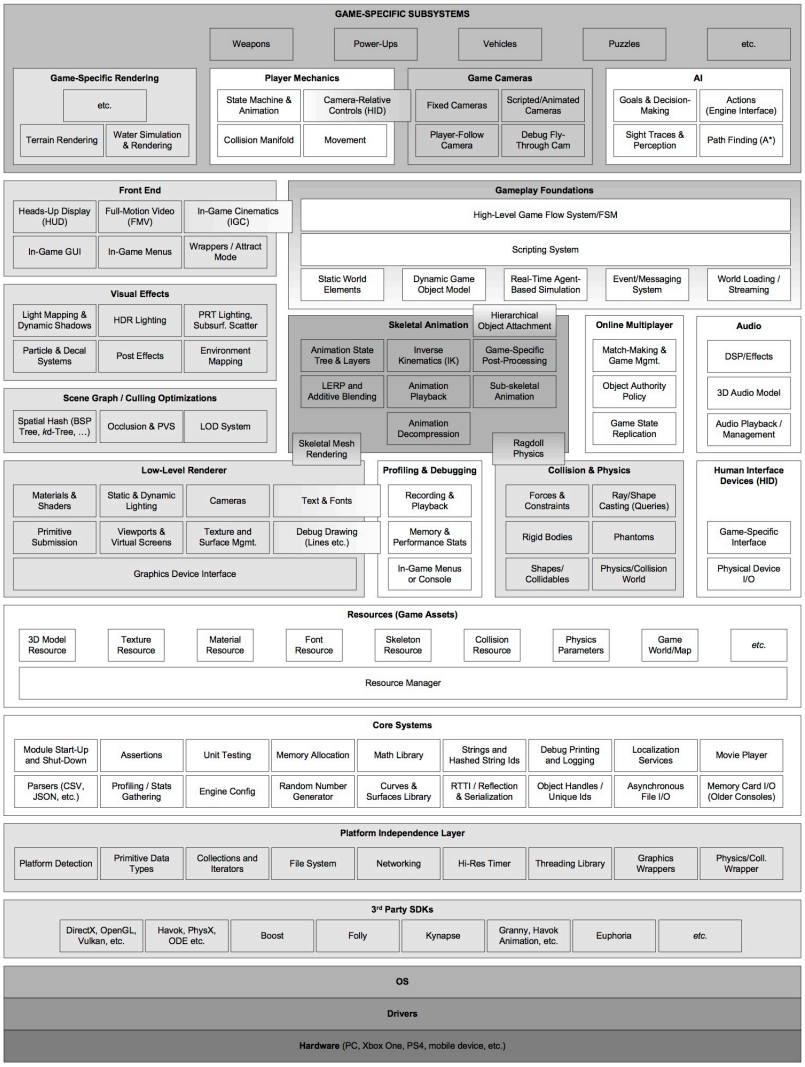

图 1: 引擎架构图[3]

这个原版的图片中的模块与功能过于完整与强大，有很多是我没有必要、也没有办法去实现的部分。经过简化后，我的引擎架构如下图所示：

[2] https://www.gameenginebook.com/index.html

[3] https://www.gameenginebook.com/figures.html

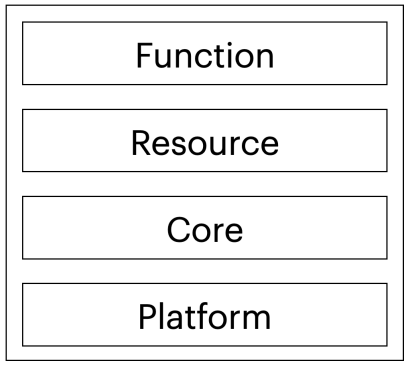

图 2: 引擎架构图[4]

该框架主要分为平台层、核心层、资源层、功能层与编辑器层。之后的各个章节将会按照分层的结构来组织。

0.2.1 基础系统

这个部分用于介绍引擎底层系统的设计，作为学习的必要过程，本项目中将跟随开发进度不断修改引擎底层结构的设计。许多基础的数据结构与算法也会出现在本节。

0.2.2 数学与几何

随后的渲染与物理部分，都与数学紧密相关。数学构成了渲染与物理模块的基石。本节主要介绍一些基本的数学内容，作为后续部分的基础。

0.2.3 渲染系统

为了能够展示场景的运行效果，需要有直观的方式展现内容，3D 渲染就是最好的方式。这个部分主要介绍渲染的相关内容，包括一个软渲染器以及对应的 GPU 版本渲染器。

0.2.4 物理系统

本节展示一个简单的物理仿真模块，用于刚体碰撞与刚体动力学仿真。为了引擎具有物理仿真的基础功能，也为了更好的表达引擎之中的交互，引入了物理模块作为底层。

0.2.5 场景管理系统

这个部分用于介绍较为大型的场景优化所需的场景管理模块的相关内容。

0.2.6 脚本及 AI 系统

本节主要介绍如何集成一个脚本系统，与场景部分相结合。由脚本系统的功能扩展 AI 的功能。

第二部分

基础模块

第一章　从零开始

1.1　第一个简单项目

作为笔记的第一部分，本节主要介绍从零开始搭建一个小型项目的过程。大部分的 C++ 项目都需要选择一个基本的编译配置，而 CMake 是其中一个流行的选择。从 github 看去，很多国内外大型的开源项目都选择了 CMake 进行项目的配置和管理。前几年我也是 CMake 的推荐者，但学习过 CMake 的我仍然觉得 CMake 过于死板，需要学习额外的 DSL（Domain Specific Language），学习曲线比较陡峭。经过查找，我发现了一个国产的开源项目 xmake[1]，可以用 Lua 编写项目配置文件，符合 C++ 编译过程的直觉，且可以通过 xmake 避免复杂繁琐的外部库安装配置过程，让人感觉很舒心。最终本项目决定使用 xmake 作为项目配置软件。

当然，本项目从熟悉的 Hello World 开始，一步步搭建起项目框架。首先是函数主体：

```cpp
#include <iostream>

int main() {
    std::cout << "Hello World" << std::endl;
    return 0;
}
```

main.cpp

这个就是简单的标准的 Hello World，通过 xmake 可以简单的配置项目：

```
target("hello_world")
    set_kind("binary")
    add_files("Engine/*.cpp")
```

[1] https://xmake.io

```
4 target_end()
```

xmake.lua

在命令行输入xmake后，就可以自动编译项目，输入xmake r hello_world后，就可以运行对应项目。通过这种流程，可以进行快速的项目编译与运行测试。

在后续章节中，若是对应的模块代码过于冗长，我将使用伪代码的形式提供运行的主要流程，对于的代码可以在 github 仓库中查看。

1.2　内存检测模块

作为第一个编写的模块，我选择了内存检测的模块。该模块的目的是提供一个跨平台，线程安全的内存泄漏检测器。参考了 Vorbrodt 的 Blog[2]，对其添加了线程安全锁后，实现了一个小型的内存检测器。由于通过宏定义完成了内存检测器的行数输出，所以该检测器只能用于普通的 new 和 delete，对于 placement new 就无力处理了。若想要不用宏定义也可以获取代码调用位置，可能就只能等待 C++20 的 std::source_location 的功能了。

该内存检测模块主要通过检测 new 与 delete 的配对问题来检测内存泄漏。一次 new 对应一次 delete，一次 new[] 对应 delete[]，避免错配。该模块使用了 set 来存储信息，在 new 的过程中插入信息，delete 的过程中检测并删除对应 new 的信息，达到匹配的效果。为了能够在程序退出时自动输出内存泄漏检查信息，定义了一个 *dump_all* 的全局变量，当程序退出时，该变量自动析构，执行最后的内存泄漏检测并输出结果。

```
1  #pragma once
2
3  // #ifdef new
4  // #undef new
5  // #endif
6
7  #ifdef RK_MEMORY_CHECK
8
9  #include <new>
10 #include <memory>
11 #include <vector>
```

[2]https://vorbrodt.blog/2021/05/27/how-to-detect-memory-leaks/

```
12 #include <utility>
13 #include <functional>
14 #include <unordered_set>
15 #include <iostream>
16 #include <ostream>
17 #include <cstdlib>
18
19 namespace Rocket::Memory::detail {
20     using string_t = const char*;
21     struct new_entry_t;
22 }
23
24 void* operator new (std::size_t n);
25 void* operator new [] (std::size_t n);
26 void operator delete (void* ptr) noexcept;
27 void operator delete [] (void* ptr) noexcept;
28
29 void* operator new (std::size_t n, Rocket::Memory::detail::new_entry_t&& entry);
30
31 void* operator new (std::size_t n, Rocket::Memory::detail::string_t file, int
       line, Rocket::Memory::detail::string_t func);
32 void* operator new [] (std::size_t n, Rocket::Memory::detail::string_t file, int
       line, Rocket::Memory::detail::string_t func);
33 void operator delete (void* ptr, Rocket::Memory::detail::string_t, int, Rocket::
       Memory::detail::string_t) noexcept;
34 void operator delete [] (void* ptr, Rocket::Memory::detail::string_t, int, Rocket
       ::Memory::detail::string_t) noexcept;
35
36 // #ifndef new
37 // #define new new(__FILE__, __LINE__, __proc__)
38 // #endif
39 #endif
```

MemoryCheck.h

```
1 #include "Memory/MemoryCheck.h"
2
3 #ifdef RK_MEMORY_CHECK
4 #include <thread>
5 #include <mutex>
```

```
6
7   // #ifdef new
8   // #undef new
9   // #endif
10
11  namespace Rocket::Memory::detail {
12      template<typename T>
13      struct malloc_allocator_t : std::allocator<T> {
14
15          T* allocate(std::size_t n) {
16              T* ptr = (T*)std::malloc(n * sizeof(T));
17              if(!ptr) throw std::bad_alloc();
18              return ptr;
19          }
20
21          void deallocate(T* ptr, std::size_t) { std::free(ptr); }
22      };
23
24      struct alignas(8) new_entry_t {
25          new_entry_t(void* p = nullptr, bool a = false, std::size_t b = 0,
26              string_t f = "N/A", int l = -1, string_t fn = "N/A")
27          : bytes{ b }, ptr{ p }, file{ f }, func{ fn }, line{ l }, is_array{ a }
                {}
28
29          std::size_t bytes;      // 8
30          void*       ptr;        // 4
31          string_t    file;       // 4
32          string_t    func;       // 4
33          int         line;       // 4
34          bool        is_array;   // 1
35          bool        padding[7]; // 7
36      };
37
38      inline std::ostream& operator << (std::ostream& os, const new_entry_t& entry)
            {
39          os << entry.bytes << "B leaked using '" << (entry.is_array ? "new[]" : "
                new")
40              << "' -> '" << entry.file << ":" << entry.line << "' in '" << entry.
                    func << "'";
41          return os;
```

```
42      }
43
44      inline bool operator == (const new_entry_t& lhs, const new_entry_t& rhs) {
            return lhs.ptr == rhs.ptr; }
45
46      struct new_entry_hash_t : std::hash<void*> {
47          using base = std::hash<void*>;
48          std::size_t operator()(const new_entry_t& entry) const { return base::
                operator()(entry.ptr); }
49      };
50
51      using new_entry_set_t = std::unordered_set<new_entry_t, new_entry_hash_t, std
            ::equal_to<new_entry_t>, malloc_allocator_t<new_entry_t>>;
52      using new_entry_list_t = std::vector<new_entry_t, malloc_allocator_t<
            new_entry_t>>;
53
54      // use local static object to store info
55      inline auto get_new_entry_set() {
56          static new_entry_set_t* new_entry_set = []() {
57              void* raw = std::malloc(sizeof(new_entry_set_t));
58              if(!raw) throw std::bad_alloc();
59              return new (raw) new_entry_set_t;
60          }();
61          return new_entry_set;
62      }
63
64      // use local static object to store info
65      inline auto get_mismatch_list() {
66          static new_entry_list_t* mismatch_list = []() {
67              void* raw = std::malloc(sizeof(new_entry_list_t));
68              if(!raw) throw std::bad_alloc();
69              return new (raw) new_entry_list_t;
70          }();
71          return mismatch_list;
72      }
73
74      // remove previous new object in info list
75      inline void operator_delete(void* ptr, bool array_delete) noexcept {
76          static std::recursive_mutex operator_delete_lock;
77          std::scoped_lock guard{ operator_delete_lock };
```

```
78          auto it = get_new_entry_set()->find(ptr);
79          if(it != get_new_entry_set()->end()) {
80              if(it->is_array == array_delete) {
81                  get_new_entry_set()->erase(it);
82              }
83              else {
84                  try { get_mismatch_list()->push_back(*it); }
85                  catch(...) {}
86                  get_new_entry_set()->erase(it);
87              }
88          }
89          std::free(ptr);
90      }
91  }
92
93  namespace Rocket::Memory {
94      inline bool dump_leak() {
95          if(auto leaks = detail::get_new_entry_set(); !leaks->empty()) {
96              std::cerr << "\n*****************************\n";
97              std::cerr << "*** MEMORY LEAK(S) FOUND ***\n";
98              std::cerr << "*****************************\n\n";
99
100             for(auto& entry : *leaks)
101                 std::cerr << entry << "\n";
102
103             std::cerr << "\n";
104
105             return false;
106         }
107         else {
108             return true;
109         }
110     }
111
112     inline bool dump_mismatch() {
113         if(auto mismatches = detail::get_mismatch_list(); !mismatches->empty()) {
114             std::cerr << "\n**************************\n";
115             std::cerr << "*** NEW/DELETE MISMATCH ***\n";
116             std::cerr << "**************************\n\n";
117
```

```
118             for(auto& entry : *mismatches)
119                 std::cerr << entry << ", freed using '" << (entry.is_array ? "
                        delete" : "delete[]") << "'\n";
120
121             std::cerr << "\n";
122
123             return false;
124         }
125         else {
126             return true;
127         }
128     }
129
130     inline void dump_all() {
131         bool result_1 = dump_leak();
132         bool result_2 = dump_mismatch();
133         if(result_1 && result_2) {
134             std::cerr << "\n***************************\n";
135             std::cerr << "***** NO MEMORY LEAK ******\n";
136             std::cerr << "***************************\n\n";
137         }
138     }
139 }
140
141 // create global dump variable, when program exit, it will use ~__dump_all__()
        automatically and dump all info
142 namespace { inline const struct __dump_all__ { ~__dump_all__() { Rocket::Memory::
        dump_all(); } } __dump_all_on_exit__; }
143
144 void* operator new (std::size_t n) {
145     void* ptr = std::malloc(n);
146     if(!ptr) throw std::bad_alloc();
147     return ptr;
148 }
149
150 void* operator new [] (std::size_t n) {
151     return ::operator new(n, Rocket::Memory::detail::new_entry_t{ nullptr, true,
            n, nullptr, 0, nullptr });
152 }
153
```

```
154 void* operator new (std::size_t n, Rocket::Memory::detail::string_t file, int
        line, Rocket::Memory::detail::string_t func) {
155     return ::operator new(n, Rocket::Memory::detail::new_entry_t{ nullptr, false,
            n, file, line, func });
156 }
157
158 void* operator new [] (std::size_t n, Rocket::Memory::detail::string_t file, int
        line, Rocket::Memory::detail::string_t func) {
159     return ::operator new(n, Rocket::Memory::detail::new_entry_t{ nullptr, true,
            n, file, line, func });
160 }
161
162 void* operator new (std::size_t n, Rocket::Memory::detail::new_entry_t&& entry) {
163     void* ptr = std::malloc(n);
164     if(!ptr) throw std::bad_alloc();
165     //void* ptr = ::operator new(n);
166     entry.ptr = ptr;
167     try {
168         static std::recursive_mutex operator_new_lock;
169         std::scoped_lock guard{ operator_new_lock };
170         Rocket::Memory::detail::get_new_entry_set()->insert(std::forward<decltype
                (entry)>(entry));
171     }
172     catch(...) {}
173     return ptr;
174 }
175
176 void operator delete (void* ptr) noexcept {
177     Rocket::Memory::detail::operator_delete(ptr, false);
178 }
179
180 void operator delete [] (void* ptr) noexcept {
181     Rocket::Memory::detail::operator_delete(ptr, true);
182 }
183
184 void operator delete (void* ptr, Rocket::Memory::detail::string_t, int, Rocket::
        Memory::detail::string_t) noexcept {
185     Rocket::Memory::detail::operator_delete(ptr, false);
186 }
187
```

```
188 void operator delete [] (void* ptr, Rocket::Memory::detail::string_t, int, Rocket
        ::Memory::detail::string_t) noexcept {
189     Rocket::Memory::detail::operator_delete(ptr, true);
190 }
191
192 // #warning If '__PRETTY_FUNCTION__' is undefined replace it with '__proc__'
        below. \
193 // Otherwise comment these warnings out and hope we get 'std::source_location'
        soon! \
194 // https://en.cppreference.com/w/cpp/utility/source_location/
195
196 // #ifndef new
197 // // #define new new(__FILE__, __LINE__, __FUNCTION__)
198 // #define new new(__FILE__, __LINE__, __proc__)
199 // // #define new new(__FILE__, __LINE__, __PRETTY_FUNCTION__)
200 // #endif
201 #endif
```

MemoryCheck.cpp

关于内存检测模块，后续仍有很多改进空间，首先需要的就是移除宏定义对 new 和 delete 的限制，使其能够运行调用 placement 的 new 与 delete 版本，以及可以在此添加自定义的内存管理器，进行自定义内存分配策略。在新的 C++ 中，已经有内存管理器的相关内容，在未来的看法中，可以考虑基于 C++ 标准库来实现内存管理的功能。

除此之外，还可以通过重载 new 与 delete 函数，来调用别的底层内存分配库，比如在这里利用了微软推出的 mi_malloc 库，相比系统自带的 malloc 函数，可以提供更高的性能。不过需要注意的是，在不同的模块间，若采用了不同的底层内存库，有可能导致程序运行时出差，难以排查，这点需要额外注意。

```
1  #pragma once
2  #include "Core/Declare.h"
3
4  #include <memory>
5  #include <new>
6
7  // ----------------------------------------------------------------------------
8  // This header provides convenient overrides for the new and
9  // delete operations in C++.
10 //
```

```
11 // This header should be included in only one source file!
12 //
13 // On Windows, or when linking dynamically with mimalloc, these
14 // can be more performant than the standard new-delete operations.
15 // See <https://en.cppreference.com/w/cpp/memory/new/operator_new>
16 // ------------------------------------------------------------------------
17
18 #if defined(__cplusplus)
19 #include <mimalloc.h>
20
21     void operator delete(void* p) noexcept;
22     void operator delete[](void* p) noexcept;
23
24     void* operator new(std::size_t n) noexcept(false);
25     void* operator new[](std::size_t n) noexcept(false);
26
27     void* operator new  (std::size_t n, const std::nothrow_t& tag) noexcept;
28     void* operator new[](std::size_t n, const std::nothrow_t& tag) noexcept;
29
30 #if (__cplusplus >= 201402L || _MSC_VER >= 1916)
31     void operator delete  (void* p, std::size_t n) noexcept;
32     void operator delete[](void* p, std::size_t n) noexcept;
33 #endif
34
35 #if (__cplusplus > 201402L || defined(__cpp_aligned_new))
36     void operator delete  (void* p, std::align_val_t al) noexcept;
37     void operator delete[](void* p, std::align_val_t al) noexcept;
38     void operator delete  (void* p, std::size_t n, std::align_val_t al) noexcept;
39     void operator delete[](void* p, std::size_t n, std::align_val_t al) noexcept;
40
41     void* operator new( std::size_t n, std::align_val_t al)   noexcept(false);
42     void* operator new[]( std::size_t n, std::align_val_t al) noexcept(false);
43     void* operator new  (std::size_t n, std::align_val_t al, const std::nothrow_t
           &) noexcept;
44     void* operator new[](std::size_t n, std::align_val_t al, const std::nothrow_t
           &) noexcept;
45 #endif
46 #endif
```

MemoryDefine.h

```
#include "Core/MemoryDefine.h"

#if defined(__cplusplus)
    void operator delete  (void* p) noexcept            { mi_free(p); };
    void operator delete[](void* p) noexcept            { mi_free(p); };
    void* operator new  (std::size_t n) noexcept(false) { return mi_new(n); }
    void* operator new[](std::size_t n) noexcept(false) { return mi_new(n); }

    void* operator new  (std::size_t n, const std::nothrow_t& tag) noexcept { (void)(tag); return mi_new_nothrow(n); }
    void* operator new[](std::size_t n, const std::nothrow_t& tag) noexcept { (void)(tag); return mi_new_nothrow(n); }

#if (__cplusplus >= 201402L || _MSC_VER >= 1916)
    void operator delete  (void* p, std::size_t n) noexcept { mi_free_size(p,n); };
    void operator delete[](void* p, std::size_t n) noexcept { mi_free_size(p,n); };
#endif

#if (__cplusplus > 201402L || defined(__cpp_aligned_new))
    void operator delete  (void* p, std::align_val_t al) noexcept { mi_free_aligned(p, static_cast<size_t>(al)); }
    void operator delete[](void* p, std::align_val_t al) noexcept { mi_free_aligned(p, static_cast<size_t>(al)); }
    void operator delete  (void* p, std::size_t n, std::align_val_t al) noexcept { mi_free_size_aligned(p, n, static_cast<size_t>(al)); };
    void operator delete[](void* p, std::size_t n, std::align_val_t al) noexcept { mi_free_size_aligned(p, n, static_cast<size_t>(al)); };

    void* operator new  (std::size_t n, std::align_val_t al) noexcept(false) { return mi_new_aligned(n, static_cast<size_t>(al)); }
    void* operator new[](std::size_t n, std::align_val_t al) noexcept(false) { return mi_new_aligned(n, static_cast<size_t>(al)); }
    void* operator new  (std::size_t n, std::align_val_t al, const std::nothrow_t &) noexcept { return mi_new_aligned_nothrow(n, static_cast<size_t>(al)); }
    void* operator new[](std::size_t n, std::align_val_t al, const std::nothrow_t &) noexcept { return mi_new_aligned_nothrow(n, static_cast<size_t>(al));
```

```
        }
#endif
#endif
```

MemoryDefine.cpp

1.3　打开第一个窗口

当然作为一个 3D 程序，需要通过显示器才能展示我们的工作成果。不同的系统有不同的显示 API，为了保证本项目的跨平台能力，需要尽可能一致的为不同平台提供相同的封装，在这里本项目选择了常用的 GLFW 库[3]。

本项目有两套渲染器，一个是 Vulkan 渲染器，一个是软渲染器。软渲染器在 CPU 中渲染完图片后，使用 OpenGL 进行屏幕渲染输出，该软渲染器主要参考了 GAMES102 课程的代码[4]。根据 LearnOpenGL 教程[5]，可以很容易写出第一个窗口程序：

```
#include "Memory/MemoryCheck.h"

#define GLFW_INCLUDE_NONE
#include <GLFW/glfw3.h>
#include <glad/glad.h>

#include <iostream>

void processInput(GLFWwindow *window);

int main() {
    glfwInit();
    glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 3);
    glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 3);
    glfwWindowHint(GLFW_OPENGL_PROFILE, GLFW_OPENGL_CORE_PROFILE);
    glfwWindowHint(GLFW_RESIZABLE, false);
#if defined(RK_MACOS)
    glfwWindowHint(GLFW_OPENGL_FORWARD_COMPAT, GL_TRUE);
#endif
```

[3] https://www.glfw.org/
[4] https://games-cn.org/games102/
[5] https://learnopengl-cn.github.io/

```
20
21      GLFWwindow* window = glfwCreateWindow(1280, 720, "Rocket", NULL, NULL);
22      if (window == NULL) {
23          std::cout << "Failed to create GLFW window" << std::endl;
24          glfwTerminate();
25          return -1;
26      }
27      glfwMakeContextCurrent(window);
28
29      if (!gladLoadGLLoader((GLADloadproc)glfwGetProcAddress)) {
30          std::cout << "Failed to initialize GLAD" << std::endl;
31          return -1;
32      }
33
34      while (!glfwWindowShouldClose(window)) {
35          processInput(window);
36
37          glClearColor(0.2f, 0.3f, 0.3f, 1.0f);
38          glClear(GL_COLOR_BUFFER_BIT);
39
40          glfwSwapBuffers(window);
41          glfwPollEvents();
42      }
43
44      glfwDestroyWindow(window);
45      glfwTerminate();
46
47      return 0;
48  }
49
50  void processInput(GLFWwindow *window)
51  {
52      if(glfwGetKey(window, GLFW_KEY_ESCAPE) == GLFW_PRESS)
53          glfwSetWindowShouldClose(window, true);
54  }
```

glfw.cpp

之后开展的渲染学习项目，将从这段程序开始扩展，并接入软渲染器来实现场景初步渲染的能力。

1.4　日志输出

1.4.1　为什么需要日志输出

在程序运行过程中，除了编译器带给我们的单步调试功能，开发过程中也需要额外的信息输出方式，用于快速检查程序输出内容是否正常以及快速定位可能存在 bug 的代码位置。比如输出 log 日志可以快速检测各个模块初始化是否按照预定顺序执行，是否全部成功初始化；在调用函数的过程中，可以检测运行逻辑是否与预期一样。更重要的是，在发布出去的软件中，我们对其进行错误追踪的方式可能只有日志输出一个渠道，所以有一个完善的日志输出模块，对我们的开发工作会有很大帮助。为了达到这个目的，本节主要介绍日志输出的部分。

1.4.2　封装日志模块

日志输出模块首先需要保证系统性能，其次可以按照不同的日志等级输出信息，便于分类，再次则需要保证多线程输出信息的安全，最后则是能够输出到日志文件等功能。经过挑选，本项目采用了 spdlog 作为日志输出的底层库，并对其进行了简单的封装。主要包括 *Log.h*[64] 与 *Log.cpp*[66] 两个文件。在程序运行开始需要初始化 Log，在项目终止时需要终止日志输出（虽然可以依靠自动析构，但是显式调用可以使流程更加清晰）。利用宏定义，可以快速添加新的日志分类。

```
1  #pragma once
2  #ifdef RK_CONSOLE_LOG
3  #include <memory>
4  #include <unordered_map>
5  #include <spdlog/spdlog.h>
6
7  #define INIT_LOG_CHANNEL(x) s_##x##_logger_ = spdlog::stdout_color_mt(#x);\
8      SetLevel(level, s_##x##_logger_.get());
9  #define END_LOG_CHANNEL(x) s_##x##_logger_.reset();
10 #define DECLARE_LOG_CHANNEL(x) \
11     public:\
12         inline static spdlog::logger* Get##x##Logger() { \
13             return s_##x##_logger_.get(); } \
14     private:\
15         static std::shared_ptr<spdlog::logger> s_##x##_logger_;
16 #define IMPLEMENT_LOG_CHANNEL(x) std::shared_ptr<spdlog::logger>\
17     Rocket::Log::s_##x##_logger_;
18 #endif
19
```

```
20 namespace Rocket {
21     enum class LogLevel {
22         TRACE = 0, INFO, WARN, ERR, CRITICAL,
23     };
24 
25     class Log {
26     public:
27         static void Init(LogLevel level = LogLevel::TRACE);
28         static void End();
29 #ifdef RK_CONSOLE_LOG
30         //     Log    μ
31         DECLARE_LOG_CHANNEL(Core);
32         DECLARE_LOG_CHANNEL(Window);
33         DECLARE_LOG_CHANNEL(Render);
34         DECLARE_LOG_CHANNEL(Event);
35         DECLARE_LOG_CHANNEL(File);
36         DECLARE_LOG_CHANNEL(Audio);
37         DECLARE_LOG_CHANNEL(App);
38 #endif
39     };
40 } // namespace Rocket
41 
42 #ifdef RK_CONSOLE_LOG
43 
44 #define RK_CRITICAL(x, ...) \
45     do{::Rocket::Log::Get##x##Logger()->critical(__VA_ARGS__);}while(0);
46 #define RK_ERROR(x, ...) \
47     do{::Rocket::Log::Get##x##Logger()->error(__VA_ARGS__);}while(0);
48 #define RK_WARN(x, ...) \
49     do{::Rocket::Log::Get##x##Logger()->warn(__VA_ARGS__);}while(0);
50 #define RK_INFO(x, ...) \
51     do{::Rocket::Log::Get##x##Logger()->info(__VA_ARGS__);}while(0);
52 #define RK_TRACE(x, ...) \
53     do{::Rocket::Log::Get##x##Logger()->trace(__VA_ARGS__);}while(0);
54 
55 #else
56 
57 #define RK_CRITICAL(x, ...)
58 #define RK_ERROR(x, ...)
59 #define RK_WARN(x, ...)
```

```
#define RK_INFO(x, ...)
#define RK_TRACE(x, ...)

#endif
```

Log.h

```
#include "Log/Log.h"

#ifdef RK_CONSOLE_LOG

// #define SPDLOG_FMT_EXTERNAL
#include <spdlog/spdlog.h>
#include <spdlog/async.h>
#include <spdlog/sinks/stdout_color_sinks.h>
#include <spdlog/sinks/basic_file_sink.h>
#include <spdlog/fmt/ostr.h>
#include <spdlog/fmt/bin_to_hex.h>
#include <spdlog/fmt/chrono.h>
#include <spdlog/fmt/fmt.h>

IMPLEMENT_LOG_CHANNEL(Core);
IMPLEMENT_LOG_CHANNEL(Window);
IMPLEMENT_LOG_CHANNEL(Render);
IMPLEMENT_LOG_CHANNEL(Event);
IMPLEMENT_LOG_CHANNEL(File);
IMPLEMENT_LOG_CHANNEL(Audio);
IMPLEMENT_LOG_CHANNEL(App);
#endif

namespace Rocket {
#ifdef RK_CONSOLE_LOG
    static void SetLevel(LogLevel level, spdlog::logger* logger) {
        switch(level) {
        case LogLevel::TRACE:
            logger->set_level(spdlog::level::trace); break;
        case LogLevel::INFO:
            logger->set_level(spdlog::level::info); break;
        case LogLevel::WARN:
            logger->set_level(spdlog::level::warn); break;
```

```
34          case LogLevel::ERR:
35              logger->set_level(spdlog::level::err); break;
36          case LogLevel::CRITICAL:
37              logger->set_level(spdlog::level::critical); break;
38          }
39      }
40
41      void Log::Init(LogLevel level) {
42          spdlog::set_pattern("%^[%l%$][%T][%n] %v%$");
43          INIT_LOG_CHANNEL(Core);
44          INIT_LOG_CHANNEL(Window);
45          INIT_LOG_CHANNEL(Render);
46          INIT_LOG_CHANNEL(Event);
47          INIT_LOG_CHANNEL(File);
48          INIT_LOG_CHANNEL(Audio);
49          INIT_LOG_CHANNEL(App);
50      }
51
52      void Log::End() {
53          END_LOG_CHANNEL(Core);
54          END_LOG_CHANNEL(Window);
55          END_LOG_CHANNEL(Render);
56          END_LOG_CHANNEL(Event);
57          END_LOG_CHANNEL(File);
58          END_LOG_CHANNEL(Audio);
59          END_LOG_CHANNEL(App);
60      }
61  #else
62      void Log::Init(LogLevel level) {}
63      void Log::End() {}
64  #endif
65  }
```

Log.h

1.5　模块划分

为了更加明确各个部分的功能，这个小节主要描述模块通用的接口，不同的功能根据具体模块进行扩展。在此引入了 IRuntimeModule 这个基类，所有的模块均继承自这个

类，保证了各个模块接口的统一。

```cpp
#pragma once
#include <string>
#include <ostream>

namespace Rocket {
    class IRuntimeModule {
    public:
        virtual ~IRuntimeModule() = default;

        // Return == 0 : everything OK
        // Return != 0 : something wrong
        [[nodiscard]] virtual int Initialize() = 0;
        virtual void Finalize() = 0;
        virtual void Tick(double step) = 0;
        // For Debug
        [[nodiscard]] virtual inline std::string ToString() const { return
            GetName(); }
    protected:
        [[nodiscard]] virtual inline const char* GetName() const = 0;
    };

    inline std::ostream& operator << (std::ostream& os, const IRuntimeModule& r)
        {
        return os << r.ToString();
    }
}

#define RUNTIME_MODULE_TYPE(type) virtual const char* GetName() const override {
    return #type; }
```

IRuntimeModule.h

模块运行的流程如下所示：

Algorithm 1 Main Loop

procedure MAIN LOOP
    Init Modules
    Init Timer
    while *isRunning* **do**

```
        Update Timer
        Tick Module
    Finalize Modules
```

第二章　基本算法

第三章　设计模式

第四章　文件系统

4.1　ļ　¼

第五章　内存管理

第六章　多线程相关

第七章　渲染

第八章　物理

第九章　脚本系统

附录 A　附录

后记